REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	27 Dec 95	

4. TITLE AND SUBTITLE
Hierarchical Holographic Modeling For Software ~~Acquisit~~ Acquisition Risk Assessment and Management

5. FUNDING NUMBERS

6. AUTHOR(S)
Richard Maury Schooff

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Students Attending:
University of Virginia

8. PERFORMING ORGANIZATION REPORT NUMBER
95-026D

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
DEPARTMENT OF THE AIR FORCE
AFIT/CI
2950 P STREET, BLDG 125
WRIGHT-PATTERSON AFB OH 45433-7765

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW AFR 190-1
Distribution Unlimited
BRIAN D. Gauthier, MSgt, USAF
Chief Administration

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

19960104 136

14. SUBJECT TERMS	15. NUMBER OF PAGES 158 +
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

HIERARCHICAL HOLOGRAPHIC MODELING FOR SOFTWARE ACQUISITION RISK ASSESSMENT AND MANAGEMENT

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy (Systems Engineering)

by

Richard Maury Schooff

January 1996

Dissertation Advisor:

Dr. Yacov Y. Haimes

APPROVAL SHEET

This dissertation is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy (Systems Engineering).

Richard M. Schooff (Author)

This dissertation has been read and approved by the Examining Committee:

Yacov Y. Haimes (Advisor)

William T. Scherer (Chair)

Douglas J. White

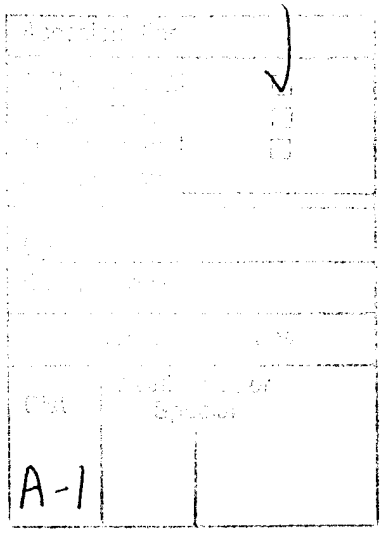
William A. Wulf

Barry W. Johnson

Clyde G. Chittister

Accepted for the School of Engineering and Applied Science:

Dr. Richard W. Miksad (Dean)



ABSTRACT

Hierarchical Holographic Modeling for Software Acquisition Risk Assessment and Management

Software has come to play an increasingly dominant role in today's world. System design, quality, and risk are predicated, as never before, on the software component of the system. Unfortunately, this significant increase in our dependence on software, which in turn causes an increase in the amount of software required, has not been matched by a corresponding increase in our capability to manage its acquisition and development. Cases of software acquisition mismanagement that resulted in large cost overruns and schedule delays have been widely reported. Maturing the capabilities of the software acquisition community require the development of appropriate tools and methodologies for risk-based decision-making management.

This dissertation addresses the assessment and management of risks associated with the software acquisition processes from a holistic perspective using hierarchical holographic modeling (HHM). The multiple visions and perspectives within which the life cycle of software acquisition is stated and modeled, provide a comprehensive framework for risk assessment and management of software acquisition. In particular, widely used models in software acquisition such as the COCOMO model, can now be extended to incorporate probabilistic as well as dynamic dimensions. The ultimate contributions of this dissertation can be found in at least two major areas: (a) in the theoretical and methodological domain of systems modeling in the quest of a more quantitative risk assessment and management framework, and (b) in advancing the state of practice in the assessment and management of software acquisition by extending highly used models in practice to incorporate more realistic probabilities and dynamic dimensions.

A probabilistic, multiobjective approach to software estimation that focuses on the risk of extreme events and utilizes the conditional expected value as an additional risk management decision-making metric is developed. Motivated by the software community's transition towards a prototype or spiral development process paradigm, a dynamic software estimation model is developed that is particularly suited for modern software development processes. The dynamic model permits analysis of the impact of management control policies on future decision opportunities, while accounting for changes over time in the development environment and the system requirements. A software estimation updating scheme is developed as an extension of the dynamic software estimation model to account for the differences between actual project resource requirements and the estimates of those requirements, and to update the overall resource requirement projections.

The HHM framework is extended to formulate software acquisition as a hierarchical decision problem. Software acquisition management decision options do not fall entirely in the domain of any single participant community, yet each party is affected by the other's decisions. HHM provides the necessary insight and coordination structure for resolving the competing issues, objectives, and decision opportunities of the several participant communities as they impact the project's cost and schedule.

Acknowledgments

I give my thanks, and am indebted to the U.S. Air Force, Colonel Daniel W. Litwhiler, and the Department of Mathematical Sciences, U.S. Air Force Academy, for their confidence and support in providing me this academic opportunity.

I am grateful for the untiring guidance of my advisor and mentor, Dr. Yacov Y. Haimes. His wisdom, insight, and gentle encouragement made this research possible. Dr. Haimes exemplifies the Jeffersonian scholar -- an active contributor to scholarship, instruction, the University institution, religious and civic affairs, as well as national and international organizations and activities. In fact, his life epitomizes the sense of holism espoused in his research. I feel privileged to have been associated with him these past years.

I have also benefited from my working relation with the other graduate students and associates of the Center for Risk Management of Engineering Systems, with particular thanks to Vijay Tulsiani and Jim Lambert. A special thanks goes to Sharon Gingras for her willing assistance, encouragement, and sense of humor, not to mention bottomless candy jar.

Above all, I thank my wife, Sharon, for her love, strength, encouragement, and support. As with all of our life, this adventure has been a joint effort and we have completed it together. Finally, my love and thanks to Ashley, Kaitlyn, and Jayson, who truly bring joy and meaning to life, and who daily remind me of my ultimate priorities.

Table of Contents

List of Tables

List of Figures

List of Symbols

a	constant
a	low parameter of the triangular distribution
b	constant
b	high parameter of the triangular distribution
c	constant
c	most likely parameter of the triangular distribution
$c(\bullet)$	environment factor in estimation models that varies over time
d	constant
e	Pressman's (weighted average) expected value
e_i	effort multiplier associated with the i^{th} attribute
E[•]	expected value
E[•|*]	expected value given the condition *
EAF	effort adjustment factor of the Intermediate COCOMO model
f(•)	probability density function
F(•)	cumulative distribution function
f_1	management control policy implementation cost
f_2	high-probability, low-consequence conditional expected value
f_3	intermediate probability and consequence expected value
f_4	low-probability, high-consequence conditional expected value
f_5	unconditional expected value
f_1^k	control policy cost at stage k
f_4^k	conditional expected value at stage k
f_5^k	expected value at stage k

$\boldsymbol{f}^{\alpha}$	α decomposition objectives of the hierarchical decision problem
$\boldsymbol{f}^{\alpha}$	β decomposition objectives of the hierarchical decision problem
f_i^{α}	the i^{th} objective of the α decomposition
f_i^{β}	the i^{th} objective of the β decomposition
$\boldsymbol{g}^{\alpha}$	α decomposition constraints of the hierarchical decision problem
$\boldsymbol{g}^{\beta}$	β decomposition constraints of the hierarchical decision problem
i	integer index
k	discrete stages (decision points) of the system
k_i	cost-per-man-month multiplier
K	cost multiplier
KLOC	thousands of lines of code
L(•)	Lagrange function
m	Pressman's most likely parameter
m	number of observations from Monte Carlo simulation
M	man-months of development effort (abbreviated notation), static model
$M^{(\bullet)}$	man-month estimate from Monte Carlo simulation, static model
M_{β}	set of Monte Carlo simulation outcomes exceeding damage partition value, static model
MM	man-months of development effort
MTTF	mean time to failure
n	integer number of iterations in a Monte Carlo simulation
o	Pressman's optimistic parameter
p	Pressman's pessimistic parameter
p	profit percentage factor associated with a cost-plus contract
PF	productivity factor
Q(•)	unreliability function
r	interest rate

R	requirements change requests
R(•)	reliability function
t_D	project development time
$u(\bullet)$	resource allocation and acquisition strategy control policy
$v(\bullet)$	random variable accounting for observation noise
$w(\bullet)$	random variable accounting for process noise
x_i	i^{th} decision variable
$\boldsymbol{x}^{\alpha}$	α decomposition decision variables of the hierarchical decision problem
$\boldsymbol{x}^{\beta}$	β decomposition decision variables of the hierarchical decision problem
$\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$	reaction of the β decomposition decision makers, given $\boldsymbol{x}^{\alpha}$
$x(\bullet)$	state of the system (interpreted as KLOC in dynamic models)
$\hat{x}(\bullet)$	deterministic representation of $x(\bullet)$
X^{α}	α decomposition definition set of the hierarchical decision problem
X^{β}	β decomposition definition set of the hierarchical decision problem
$y(\bullet)$	effort (cost) output
$\hat{y}(\bullet)$	deterministic representation of $y(\bullet)$
$Y^{(\bullet)}$	man-month estimate from Monte Carlo simulation, nonlinear dynamic model
Y_{β}	set of Monte Carlo simulation outcomes exceeding damage partition value, nonlinear dynamic model
z(•)	instantaneous failure rate function
α	partitioning probability for the risk of extreme events in the PMRM
α	α decomposition sub-problem of the hierarchical decision problem
β	damage axis partitioning of the risk of extreme events in the PMRM
β	β decomposition sub-problem of the hierarchical decision problem
ε_i	tolerance value of the i^{th} constraint in the ε-constraint method
λ	Lagrange multiplier

λ_{ik}^{jl}	tradeoff rate between objectives f_i^j and f_k^l
μ	mean value
σ^2	variance

Chapter 1
Introduction and Problem Background

1.1 Motivation for the Research

Software has come to play an increasingly dominant role in today's world. It controls the way organizations operate, aids in analyzing and determining corporate strategies, and generally helps organizations operate more efficiently. As computer usage has become central to organizational activities and engineering system design, the software component of these systems has become increasingly important. Software has assumed the systems integration role -- components that before could not be integrated, or were interconnected through hardware means, are now linked via software. Systems quality is predicated, as never before, upon the quality of its software. System risk is increasingly being defined relative to the risk associated with the software component.

Unfortunately, this significant increase in our dependence on software, which in turn causes an increase in the amount of software required, has not been matched by a corresponding increase in our capability to manage its acquisition and development. Acquisition officials, whose training and experience previously focused on the hardware component of a system, now find themselves concentrating more of their energies, concern, and resources on the software component.

The inability to effectively manage the software component leads to project cost overrun and schedule slippage. Rothfeder [1988] gives an example in the case of Allstate Insurance which hired Electronic Data Systems in 1982 to build an $8 million computer system expected to be completed in 1987. It was finally completed in 1993 at a price of over $100 million [Fairley 1994]. Neumann [1988] gives the example of Bank of America which in 1988 abandoned a computer system originally estimated to cost $20 million in 1982 after spending over $60 million trying to get it to work. GAO [1992a] states that software development has been a major problem during the C-17 development program. Although operational C-17 aircraft are now in the Air Force inventory, the overall project is over 2 years behind schedule and $1.5 billion over its 1985 cost estimate of $4.1 billion. McFarlan [1981], Rothfeder [1988], and Neumann [1988] provide numerous other

instances where software projects have gone over budget and schedule more frequently than not.

Boehm [1989] states that these project disasters can generally be traced back to risk items that were either not identified, were improperly assessed, or improperly dealt with. For example, GAO [1992a] states that the Air Force underestimated the size and complexity of the C-17 software development effort and assumed that the software development would be low-risk without performing any analysis to support and document the assumption. Thus, it is clear that the risks in software acquisition must be identified and managed properly in order to minimize the losses resulting from such runaway projects.

Effective management of modern, complex processes such as software acquisition requires capable, mature direction. Good management of technological systems must address the holistic nature of the system in terms of its hierarchical, organizational, and functional decision making structure; the various time horizons; the multiple decision makers, stakeholders, and users of the system; and the host of technical, institutional, legal, and other socioeconomic conditions that require consideration. Maturing the capabilities of the software acquisition community will require increasingly sophisticated analytic tools and methodologies to identify program risks, evaluate their potential adverse impact, and effectively incorporate risk considerations in the decision making management framework.

1.2 Impact of the Research.

The overall aim of this research is the development of theoretical and methodological foundations upon which we can enhance software acquisition management through the development of a holistic and systemic risk assessment and management framework. The particular objectives in support of the overall focus include:

- Developing a holistic framework for software acquisition that provides a comprehensive structure to identify risk sources, assess the risks, explicitly include the consideration of uncertainty, and coordinate competing issues that dominate software acquisition decision making.

- As part of the review of literature, identify current methods, models, and tools that assist in the software estimation effort -- including software cost, schedule, and performance estimation.

- Extend current software estimation practices to include explicit consideration of the inherent uncertainties - hence, risks -- associated with a software acquisition endeavor.

- Develop a dynamic software estimation approach that is particularly suited for modern software development practices; namely, prototyping and spiral development processes.

- Devise an approach for updating the software estimates as an on-going activity, conducted throughout the life cycle.

- Demonstrate a hierarchical multiobjective decision-making framework for coordinating competing decision-making issues among software participant communities as they impact the project's cost and schedule.

The intent of this work is not to specifically address and consider all of the multiple aspects associated with the software acquisition process, but to develop a framework that would enable the consideration of such complexities and interconnectedness, and then focus the research effort on a most-critical element of the software acquisition process -- software estimation.

The research builds on the framework of risk assessment and management for engineering systems, hierarchical holographic modeling (HHM), software estimation modeling, the partitioned multiobjective risk method (PMRM), the risk of extreme events, and dynamical modeling.

While many examples and references are made to government software acquisition (due to the federal government's tremendous expenditures for software products and services and to the fact that government acquisition procedures, regulations, and results are available for public review), application of the results of this research are intended to strengthen the software acquisition program manager's (government or industry) ability to: i) identify and comprehend the complexities and risks associated with a software acquisition program, ii)

quantify the uncertainties associated with the program, and iii) make trade-off judgments for resolving competing issues and objectives.

1.3 Organization of the Dissertation

This Chapter 1 demonstrates the need for a comprehensive approach to software acquisition risk assessment and management for identifying and evaluating project risks in situations of uncertainty. Background information regarding the criticality of software for modern systems, the complexity of the software acquisition process and all it encompasses, software risks, and past trends and shortcomings in software acquisition research point to the need for this research.

Chapter 2 reviews the literature pertinent to software acquisition risk modeling. The concepts of risk, and risk assessment and management are introduced and reviewed. Model management methods, those approaches for structured consideration of multiple analytic models, are reviewed. Hierarchical holographic modeling (HHM), which provides the theoretical and methodological basis and incentive for many of the results of this dissertation, is introduced and reviewed. Software estimation and software reliability models are reviewed. An extensive review of software estimation methods and tools is provided in an appendix. The general concepts of probabilistic analysis are introduced, and an explanation of the fallacy of the expected value motivates the extension of classical approaches. The partitioned multiobjective risk method (PMRM), with its risk measure of extreme events, f_4, is introduced.

Chapter 3 develops a hierarchical holographic model for software acquisition. Exploiting the inherent synergy of HHM's descriptive and analytic duality provides the necessary theoretical, methodological, and practical foundation for a software acquisition risk assessment and management framework. Only by exploring the various dimensions and perspectives of software acquisition, and the combinations of perspective elements, can risk identification be properly accomplished.

Chapter 4 is a derivation and application of the exact solution for the expectation functions of the triangular distribution. In situations such as software acquisition, where insufficient empirical evidence or expert judgment rule-out the use of other probability distributions, analysis using triangular distributions is often desirable. Deriving risk functions for the

triangular distribution enhances decision making in conditions of uncertainty, permitting probabilistic analysis that includes the additional information of the conditional expected value. Results are deployed to software acquisition decision making situations.

Chapter 5 develops two probabilistic, multiobjective approaches for software estimation. A method for direct estimation that employs the results of Chapter 4 is developed and demonstrated. The second method, using Monte Carlo simulation, extends probabilistic analysis to the range of existing software estimation models. A methodology for evaluating the unconditional and conditional expected values of a Monte Carlo simulation is developed. The approach is deployed to the COCOMO model; comparisons of the probabilistic approach, the COCOMO model, and the actual project results are made using the original COCOMO data set.

Chapter 6 extends the current state-of-art in software cost estimation modeling by developing multistage, dynamical software cost estimation models. As the software community embraces the spiral development process model, software cost estimation models that are responsive to the new paradigm are required. No longer a single time-period activity, software cost estimation models must account for the dynamics of changing software requirements and design over multiple time periods. Applying the probabilistic cost estimation method of Chapter 5 with its multiple objective risk functions constitutes a multiple objective decision problem that is solved over multiple stages.

Chapter 7 explores two additional issues related to software estimation: i) the on-going role of software estimation for resolving the discrepancies between actual and estimated project progress, and ii) the coordinated resolution among participant communities of software program management. Actual project development effort and schedule rarely matches its estimates exactly -- appropriately adjusting the estimation models to update the eventual effort and schedule requirements is facilitated through an HHM investigation. With a revised estimate, appropriate management control policies can be selected. These policy decisions, however, are not entirely in the jurisdiction of any one participant community, but are shared among the participants. Each participant is affected by the other's decisions. Investigation of the unique and overlapping problem elements through the HHM increases understanding and provides the framework for mutually agreeable solutions.

Chapter 8 provides a summary of the contributions of the dissertation and some recommendations for future work.

Three appendicies are included. Appendix A is a detailed tutorial on the popular software estimation model, COnstructive COst Model (COCOMO). As COCOMO is widely recognized as the industry standard software estimation model, the results of this research are applied to the model. Appendix B is a thorough review of the many software estimation tools available on the market. A brief description of each tool, along with information concerning its vendor or supporting organization is provided. Appendix C includes derivations of the remaining conditional expectation functions for the triangular distribution that were not included in Chapter 4.

1.4 Software Acquisition - Background

The concept of software acquisition is in fact a misnomer (even though it is included in the title of this dissertation). In reality, government acquisition focuses on acquiring *systems*, not exclusive software per se; these systems include an increasingly major software component. Simply stated, systems acquisition (to include software acquisition) is the process and activities associated with procuring a solution system that meets a real-world, operational need. Major systems acquisition is a complicated jumble of regulations, organizations, activities, decisions, and procedures. The following sections provide background on some of the elements that contribute to the complexity associated with software acquisition: the criticality of software, the participants involved in software acquisition, the software acquisition process, software risks, and trends in software development and software acquisition research.

1.4.1 Criticality of Software in Modern Systems

As computer usage has become central to organizational activities and engineering system design, the software component of these systems has become increasingly important. The criticality of software's role in modern systems is well documented and universally accepted [Sage 1995], [Blum 1992], [GAO 1992b], [DSB 1987].

Chittister and Haimes [1994] document the occurrence of a power shift -- the transfer of importance -- from hardware to software within modern systems. Software has become the principal system design component, as well as the principal factor affecting system quality. In fact, software has been described as the "Achilles heel" of modern weapon

systems because it is a key determinant of development schedules and because the performance of key functions such as navigation, enemy detection, and fire control depends on it [GAO 1992b]. As depicted in the introductory section of this Chapter, examples of system failure whose root cause can be attributed to software failure have been well publicized.

Expenditures for software are growing rapidly. The Department of Defense "reached an annual software expenditure level in mission-critical computer systems [defined as systems whose failure would endanger life, equipment, or success of the defined task] of about $9 billion in 1985, with projections of over $30 billion annually by the mid-1990s" [DSB 1987]. As expenditures grow, so do concerns about the reliability, cost, and performance of these complex software systems.

Due to the continued expansion of software's commanding role in modern systems (and the budget for such systems), the ability to effectively acquire and integrate software into these systems will continue to be an increasingly important issue.

1.4.2 Participants in the Software Acquisition Process

There are three principal participants, or groups of participants, in an acquisition endeavor: the user, the customer, and the contractor. As depicted in Figure 1.1, under current practice the user and contractor communities generally communicate through the customer community. Quite often, an almost adversarial relationship exists among the three communities, stemming from their competing interests and objectives.

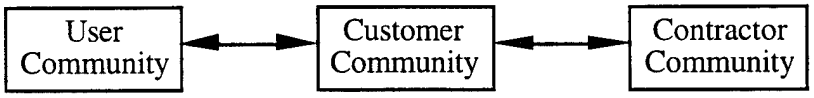


Figure 1.1 Acquisition process participants

In government as well as corporate acquisition, these groups rarely constitute single individuals, but each is often comprised of one or more organizations and their representatives. The user's role is to identify and validate operational needs; the contractor is responsible for developing a system that will satisfy the operational need; and the customer organization is responsible for accurately translating the user's needs into the contractual language of systems requirements, selecting the most appropriate system design

and the best qualified contractor, monitoring system development, accomplishing contract management and negotiation functions, and conducting system testing and acceptance.

1.4.3 The Software Acquisition Process

The *software acquisition process* encompasses all activities required to define, develop, test, and procure a software product. Acquisition starts with the process of defining requirements for a system and ends when the software is placed into operational use. Software acquisition is closely related to the term *software development* (the set of activities that results in software products) [DoD 1994]. The distinction made between the terms, as used in this dissertation, is that *software acquisition* implies the viewpoint, responsibility and focus of the customer community, while *software development* implies the activities and focus of the contractor community. Software acquisition encompasses all of the stages and activities of software development, with additional activities specific to the customer organization. Another term that is often used in connection with software acquisition or software development is the term *software life-cycle*. Again, there is a subtle distinction among the terms. The software life-cycle begins with the establishment of a mission need, encompasses all the activities of software acquisition, and includes the additional activities of operations, maintenance, reconfiguration, and retirement.

1.4.4 Phases in the Software Process

The software process discussion contained in the software literature is primarily focused on software development processes and on the contractor's role in the development process. Nonetheless, existing software process models provide the foundation for describing the central activities associated with software acquisition. An acquisition endeavor begins with the recognition that a problem or a need exists which can be solved or fulfilled by a software product. This leads to the design and implementation of the software product. Broadly, the activities in the process can be classified as [Abbott 1986]:

> **Requirements:** This phase covers the development of the system requirements and other specifications. The requirements characterize the expectations of the user from the software system and define the objectives to be achieved. These objectives may evolve over time. A detailed requirement specification could also describe the external behavior of the system.

Design: A design describes the internal organization of a system. A detailed system design would include details on the various modules, sub-modules, and the interaction required. A set of preliminary test cases to be used for system debugging and evaluation is also specified in this phase. A detailed design also usually incorporates the concepts of dataflow, components, and would include flow charts and pseudo code.

Implementation: This phase covers the actual coding of all the components of the system as well as the interactions between them. The debugging of the lower-level components of the system is usually a part of this phase.

Testing: In this phase the debugging and testing of the overall system is carried out. The debugging is performed on the interactions between the different modules and the testing of the complete system is carried out according to the system test cases and data defined earlier.

Operation: In this phase the system is fully operational and is used by the end user. Some debugging and minor revisions may be done in this phase as required.

Maintenance: In this phase, the system is modified and enhanced according to new requirements desired by the user. Usually during this phase, any debugging or minor revisions done on the system during the operation phase are cataloged and incorporated into the system documentation.

1.4.5 Models of the Software Process

Software life cycle models identify various phases and associated activities required to develop, acquire, and maintain software, and provide excellent input into the software estimation process. This section discusses some of the common models of the software development process. The intent is to highlight the complex coordination of activities, processes, and individuals in a software acquisition effort.

1.4.5.1 Waterfall Model. Traditionally, the software process has been described in terms of the waterfall model [Royce 1970], also known as the phased or the stagewise model. In this model (an adaptation from the hardware development process model), the software development process goes through each of the phases described in the previous section once. There are criteria defined for the transition from one phase to the next. At each phase, there is a feedback loop so that the product does not move on to the next phase until the validation of the previous phase is complete. The two main features of the waterfall model are the validation and verification of the current phase before progression to

the next phase and iterations of the previous phase in order to remove any remaining problems. Figure 1.2 illustrates the waterfall model.

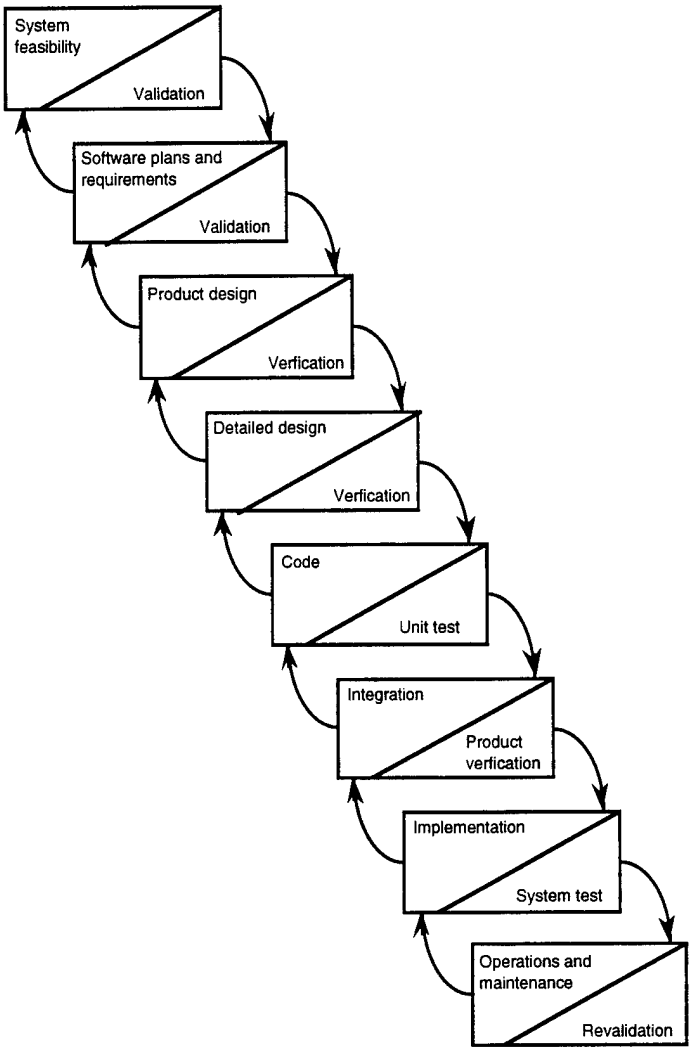


Figure 1.2 Waterfall model of the software process [Boehm 1981]

While the model presents a logical, organized approach, its inflexibility in adapting to unique requirements of modern software development has led many in the software community to feel that the model has been discredited [Blum 1992]. Furthermore, the model was never really followed because of the iterative nature of software development. Consequently, the spiral model is gaining more acceptance, at least as the iterative approach.

1.4.5.2 Prototype Model. Two prototype models, Rapid Prototyping, and Incremental Development, are discussed in [Davis et al. 1988]. The concept behind the use of the prototype model is that small problems are easier to solve than large ones. Therefore, the software development process consists of building a series of smaller systems for testing and evaluation. The experience gained from testing these prototype systems is then used to build the final system. The purpose of development of the intermediate products is to increase the understanding of the system and therefore only the most immediate issues are addressed. In most cases, the intermediate products from the application of the prototype model are discarded.

The prototyping approach provides a more realistic validation for requirements than reviewing a set of specifications and manuals, it helps in minimizing requirements changes from a long development period, and makes it possible to generate a number of alternative systems for comparative trials. However, the failure of initial prototypes may discourage the users and the developers. It can lead to the acceptance of a suboptimal system which may require substantial rework before it can be accepted.

1.4.5.3 Spiral Model. The spiral model of software development [Boehm 1988] integrates the prototype with the waterfall model as shown in Figure 1.3. The spiral model can accommodate the waterfall and the prototype models as special cases. This model consists of a series of learning cycles with each iteration including the phases of identification, evaluation, planning, and testing. With each successive iteration, greater insight is gained and system development is improved.

There are no restrictions as to the number of cycles to be followed; the process is continued until an acceptable product is developed. By combining the stages from the waterfall model and the iterative nature of the prototype model, the spiral model addresses the limitations of the waterfall model. However, it only addresses development processes where requirements are specified but not processes where requirements evolve [Carr, 1989].

1.4.5.4 Other Software Process Models. Additional software process models have been proposed, although not widely accepted. Carr's circular model [Carr 1989] was motivated by the need to address development processes where requirements evolve. This model consists of two independent cycles with three stages in each cycle. The first cycle consists of requirements, design, and evaluation functions while the second cycle consists of analysis, build, and evaluation functions. Progression from the first cycle to the second

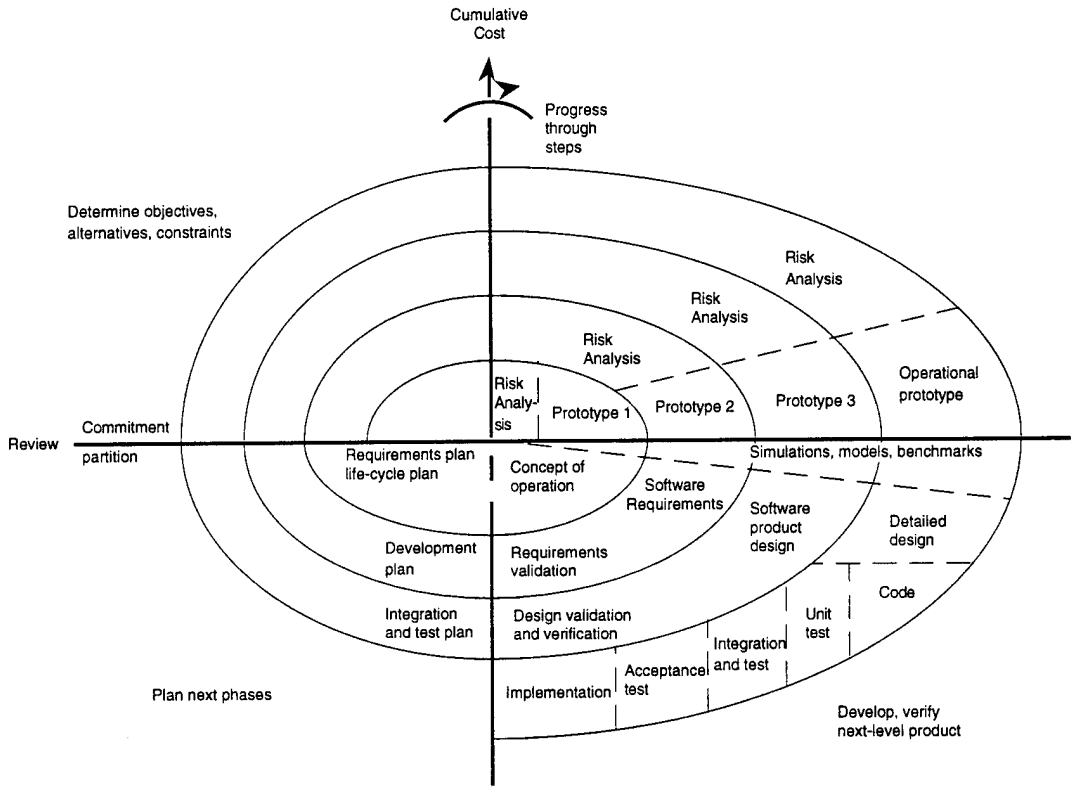


Figure 1.3 Spiral model of the software process [Boehm 1988]

is based on sucessful development of a proposed solution, otherwise, the process repeats the first cycle. The Reusable Software model and the Transform model [Davis 1988] include development phases and decision points for software projects that primarily depend on modifying existing software systems rather than developing new code.

1.4.5.5 Software Process Models and Software Acquisition. The evolutionary process models, such as the spiral model or circular model, provide a flexible structure for program managers to develop and adjust project design and objectives over time. Under such a paradigm, the program manager can iteratively evaluate alternative software designs to determine which best addresses user needs within resource constraints and at acceptable risk. Unfortunately, the government acquires software using a system known for its rigidity and dependence on the waterfall approach [DoD 1991].

One of the major initiatives associated with recent acquisition reform efforts is modifying the existing process to better meet the unique requirements of software projects. Adoption of Military Standard 2167A [DoD 1988] was an attempt to encourage modern software

development methods and flexible design opportunities in government software acquisition programs. Continued calls for improvements that include adaptive design, prototyping, and other iterative development approaches have been recommended [SST 1989], [DSB 1994], [SAB 1994]. A newly-released DoD guideline for software development, MIL-STD-498 [DoD 1994], includes options for flexible acquisition strategies.

1.4.6 Software Risk

The categories of software risk are generally divided into those that deal with the process of acquiring the software and with the product itself. Chittister and Haimes [1994] refer to these two areas as software nontechnical risk and software technical risk. *Software nontechnical risks* are associated with the programmatic aspects of the acquisition process [Haimes and Chittister 1995]. This may include risks associated with general management, contractor selection, scheduling, budgeting, etc. *Software technical risk* refers to the adverse event that the software does not meet its intended functions and performance requirements.

Charette [1989] highlights several general characteristics of the software acquisition environment that contribute to the presence of risks:

- Software development projects are *complex*. The problem elements are numerous and the interrelationships among the elements are *extremely complicated*.
- Relationships between elements may be *highly nonlinear*.
- The elements of the problem are *uncertain*.
- The situation is *dynamic*; conditions *change continuously*, equilibrium is rarely encountered.
- Software development is *a human endeavor*, with all the problems that brings.

The inherent uncertainty in software acquisition is a result of some of its key characteristics: software evolves rapidly, is difficult to explicitly define and specify, acquisition officials often lack software understanding, and there is difficulty in estimating project costs and time requirements. The current acquisition process takes an average of 16 years to field a new weapons system [Pages 1994], while software and computer life cycles are as short as 1 to 2 years. A software solution could become obsolete before being delivered.

The increasingly popular approach of using commercial off-the-shelf (COTS) packages to construct software systems rather than build all the needed components is often looked

upon as being a less-risky venture. Such buy-and-integrate strategies, however, have their own inherent risks. Some of the risks associated with COTS include [Fairley 1994]:

- Integration. Integrating the different packages' data formats and communication protocols can be tricky. Often it takes more effort to integrate packages than to build the components from scratch.
- Upgrading. The new version of a vendor's package may have a different interface or feature set than the old version. The new version may require more memory or run more slowly.
- Lack of source code. Buying a COTS package provides only the object code, making enhancing the system nearly impossible. Vendors are reluctant to provide the source code. Even if the source code is available, it is often so difficult to understand that it is hard to modify.
- Vendor failure. What happens if the vendor goes out of business or is bought out?

1.4.7 Software Development Research

Over the past three decades, much research has been conducted relative to software development practices and processes -- those generally associated with the contractor's domain of an acquisition project. Early efforts of applying and extending the practices and principles of engineering to software led to the development of the software engineering discipline (histories of the origins of software engineering are found in [Sage 1995], [[Blum 1992], and [Charette 1989]). The development of software life cycle models and software process development models [Feiler and Humphrey 1992], have helped to bring a degree of standardization and process improvement to the software development community.

Much research has focused on improving the software development process (e.g., [Humphrey and Kellner 1989], [Kellner 1991], [Heineman et al. 1994]). Business realities such as strong competition, pressure for increased profits, and external regulations have spurred the momentum for an improved software development process [Austin and Paulish 1993]. Improving the software development capabilities of software vendors by improving their software development process maturity is the focus of the Software Engineering Institute's Capability Maturity Model (CMM) [Paulk et al. 1993]. This tool "provides software organizations with guidance on how to gain control of their process for developing and maintaining software and how to evolve toward a culture of software engineering excellence" [Paulk et al. 1992]. Other related advances including software process assessment [Humphrey 1989] [Kellner and Hansen 1988], software metrics

[Shepperd 1995] [Rozum 1992] [Mills 1988], CASE tools [Brown et al. 1994] [Barros 1992] [Nejmeh 1990], software quality [Florac 1992] [Schulmeyer 1992] [IEEE 1990], and software reliability [Neufelder 1993] [Musa et al. 1990] [Glass 1979] have been accomplished predominately on behalf of the software contractor, to aid in the actual development of software.

1.4.8 Software Acquisition Research

Unfortunately, when compared to the volume of software development research, relatively little has been studied and written specifically for the customer's benefit, i.e., the development of methods and approaches to effectively manage a software acquisition effort. As it is the customer community that assumes the major role in an acquisition effort, "the next major improvement in software acquisition will come by turning the focus to the customer's role in this complex process" [Sherer and Cooper (draft) 1994].

Recent developments by the Software Engineering Institute [Ferguson et al. (draft) 1995] [Sherer and Cooper (draft) 1994] have led to an initial version of a Software Acquisition Capability Maturity Model (SA-CMM) for maturing the acquisition capabilities of the customer community. Parallel, yet independent research in this area is presented in [Baker et al. 1994]. As with the CMM, the SA-CMM is both an evaluative tool as well as an aid for increasing a community's capability. The SA-CMM (Figure 1.4) proposes a structure of five progressive levels of maturity for software acquisition capability, along with key process areas for each level. The premise of the maturity model is that increasing in capability by moving up in maturity level also increases the probability for success; a level 3 organization has a greater probability of achieving success than a level 2 organization [Ferguson et al. (draft) 1995]. Increasing the acquisition capability of the customer community improves productivity and program quality while simultaneously reducing risk.

The maturity progression is intended as an upward flow, in that satisfying the requirements of one level leads to higher-level functions. The key process areas at any given level describe the minimum requirements for that level of maturity. While a lower-level organization may be practicing some elements of a higher maturity level, it cannot achieve the higher level unless all of the requirements of all of the key process areas have been satisfied.

Level	Focus	Key Process Areas	Result
5 Optmizing	**Process Optimization**	Continuous Process Improvement Acquisition Change Management	**Productivity & Quality**
4 Managed	**Quantitative Management**	Quantitative Process Management Software Quality Management Asset Management	
3 Defined	**Integrated Project Management**	Organization Process Definition Organization Process Improvement Project Performance Management Contract Performance Management Intergroup Coordination Acquisition Risk Management Training Program	
2 Repeatable	**Stabilized Contract Management**	Acquisition Management Planning Solicitation Requirements Development Requirements Management Project Office Management Contract Tracking & Oversight Evaluation and Acceptance Transition & Maintenance	
1 Initial	**Ad hoc**		**Risk**

Figure 1.4 Software Acquisition Capability Maturity Model (SA-CMM) based on [Ferguson et al. (draft) 1995] [Sherer and Cooper (draft) 1994]

Maturity in software acquisition capability implies a verified, repeatable, effective *process* and a *quantitative management* framework for governing that process. Level 2 focuses on stabilizing the management process, "allowing project teams to repeat successful practices employed on previous projects" [Ferguson et al. (draft) 1995]. A level 3 organization employs an integrated project management, contract management, and risk management strategy. At level 4, *Quantitative Management*, the customer builds on the level 3 management framework by setting and monitoring quantitative quality goals for processes, products, and services. The highest maturity level, Optimizing -- level 5, is focused on continuous process improvement; the organization has the means to identify processes that can be optimized and has statistical evidence available to analyze process effectiveness. The general framework of the SA-CMM underscores the need for a comprehensive software acquisition risk management vision with appropriate quantitative analysis for decision-making support.

1.5 Chapter Summary

Software acquisition encompasses a wide range of activities and concerns far beyond that of the actual development of a software product. The complexity of the process now extant requires new analytical models and techniques that explicitly consider the uncertainties associated with software acquisition, and requires a multi-visionary approach to the understanding and quantification of both the complexity and the risks associated with the elements thereof.

Strengthening the software acquisition manager's ability to systematically identify project risks, quantify their impact, assess various management policy alternatives, and do so in the ever-changing, dynamic environment of software acquisition would be of benefit. Such is the intent of this research.

Chapter 2
Literature Review and Model Evaluation

Software acquisition management requires the completion of many complex data analysis and decision making activities. Operating in an uncertain environment, one of the most challenging aspects of software acquisition management is accurately determining the needed resources, required schedule, and performance measures for software development. Such a task requires establishing the functions and characteristics of the desired system, estimating the size and design of the software product to be produced, and estimating development effort requirements. Accounting for, and effectively managing software acquisition risks is a key element in the management process.

This Chapter provides a discussion of the relevant theories and analytical approaches that impact the decision making activities associated with software acquisition. This first includes a review of the concepts of risk, and risk assessment and management. Model management methods and hierarchical holographic modeling (HHM) -- approaches for considering the integration of modeling efforts -- are reviewed. This Chapter reviews the various analytical approaches associated with software estimation: effort and schedule models, and software performance models. An extensive review of software estimation tools is provided, with detailed information in an accompanying appendix. The general concepts of probabilistic analysis are introduced, and an explanation of the fallacy of the expected value motivates the extension of classical software estimation approaches. The partitioned multiobjective risk method (PMRM), with its risk measure of extreme events, f_4, is introduced.

2.1 Risk

Risk has been defined in many different ways. Some examples of these definitions are:

- Lowrance [1976] has defined risk as "a measure of the probability and severity of adverse effects."
- Rowe [1977] has defined risk as "the potential for realization of unwanted, negative consequences of an event."

- Kaplan and Garrick [1981] suggest a quantitative definition of risk as a set of triplets. This set is framed as a series of three questions:

  - What can happen? (i.e., What can go wrong?)
  - How likely is it to happen?
  - If it does happen, what are the consequences?

Risk, therefore, consists of two interrelated elements: i) the undesirable consequence or outcome, and ii) the probability or potential for the realization of that outcome. As introduced in Chapter 1, software acquisition risk includes software technical risks and software nontechnical risks.

2.1.1 Risk Assessment and Risk Management

The basic goal of *risk assessment* is to describe the current risk scenario by answering the three questions posed by Kaplan and Garrick [1981]. Risk assessment has been divided into three phases, namely, risk identification, risk estimation, and risk evaluation. The first phase, *risk identification*, is the reduction of descriptive uncertainty [Rowe 1977]. Relating to the definition of risk by Kaplan and Garrick, risk identification is the list of answers to the first question. The second phase, *risk estimation*, is the reduction of measurement uncertainty [Rowe 1977] that comes in answering the second two questions. *Risk evaluation* is the assignment of values to the probabilities and the consequences associated with the scenarios identified in the risk identification phase.

The *risk management* process builds upon the risk assessment process by seeking answers to a second set of three questions [Haimes 1991]:

- What can be done? (i.e., What options are available?)
- What are the associated trade-offs between the options in terms of costs, benefits, and risks?
- What are the future impacts of current management decisions on future options?

The answers to these questions facilitate risk-based decision making.

2.2 Model Management

Evaluation of complex systems and processes relies on the use of multiple analytic models. There has been a recognized increase in the use of computer-based management science

models to solve problems encountered in all areas of government, business, and industry [Muhanna and Pick 1994]. The requirement for, and the availability of, a large and diverse collection of decision models has prompted the growing body of work focusing on the topic of model management.

The broad view of model management is "the philosophy that: 1) models are a resource (like data) that should be managed; and 2) modeling is an ongoing activity that should be managed, integrated, and coordinated in order to avoid wasteful, suboptimal decisions" [Muhanna and Pick 1994]. During the last decade, research concerning model management has been developed along such diverse paths as database theory, artificial intelligence, and conceptual graphs.

2.2.1 Database Theory-based Model Management

The success of database management systems in overcoming the problems of data management motivated a number of researchers to investigate the use of this technology in addressing issues in model management. Such data-oriented approaches are found in [Miller and Katz 1986], [Lenard 1986], [Liang 1985], and [Stohr and Tanniru 1980]. Unfortunately, these approaches are reported at the conceptual level with little consideration given to the feasibility or practicality of actual implementation.

2.2.2 Artificial Intelligence-based Model Management

Artificial intelligence techniques for knowledge representation have also been investigated for model representation (e.g., [Dutta and Basu 1984], [Fedorowicz and Williams 1986], [Shaw et al. 1988], and [Dutta and Mitra 1993]). Integrating database and formal logic approaches was proposed in [Bonczek et al. 1981]. Use of semantic nets as a vehicle for representing knowledge about models was proposed in [Elam et al. 1980]. Other than simple prototype systems, automated model synthesis through artificial intelligence has yet to be implemented.

2.2.3 Graphical-based Model Management

Graphical approaches to model management utilize visual images to capture and describe the relationship between models [Muhanna and Pick 1994], [Muhanna 1994], [Basu and Blanning 1994]. These approaches provide a framework for model composition by

identifying models that may be combined into an integrated model [Basu and Blanning 1994]. In addition to the visualization benefits that graphical representation offers, the process of model integration is facilitated by identifying the connectivity of models that is made apparent in the graphs. Interconnecting the output of one model with the input of another, component models can be coupled together to assemble composite models. Muhanna and Pick [1994] advocate a modular, hierarchical model management approach in which composite models can be formed from the coupling of atomic models (those without components). These composite models can then be used as a component coupled with other models to form a new higher-level model.

Model management systems (MMS) that essentially employ a graphical approach have been proposed as a component of computerized decision support systems [Applegate et al. 1986], [Blanning 1985], [Sprague and Carlson 1982]. A MMS is "a software system that facilitates the development, storage, manipulation, control, and effective utilization of models" [Muhanna and Pick 1994]. While prototype MMS frameworks have been successfully developed, the inherent benefits of a graphical approach, along with ease of application, indicate that this approach has the most promise for implementation and use in actual model management applications.

2.3 Hierarchical Holographic Modeling (HHM).

Since its origin in 1981, the HHM has provided a general framework for addressing the modeling of complicated, multiple objective problems of large scale and scope [Haimes 1981]. HHM's multivisionary approach to problem definition and risk identification has been widely, although often indirectly, accepted. Throughout his book *Metasystems Methodology*, Hall [1989] uses HHM to recount the history of systems methodology, and to distinguish the varied applied systems methodologies from each other. He states [Hall 1989]:

> History becomes one model needed to give a rounded view of our subject within the philosophy of *hierarchical holographic modeling*, defined as using a family of models at several levels to seek understanding of diverse aspects of a subject and thus comprehend the whole.

Fundamentally, HHM is grounded on the premise that complex systems and processes, such as software acquisition, should be studied and modeled by more than one single

model, vision, or perspective. HHM possesses a dual nature as 1) an holistic, investigative paradigm, and 2) a mathematically-sound, hierarchical, multiple objective decision-making methodology. The HHM approach identifies and coordinates multiple, complementary decompositions of a complex system. A decomposition is a hierarchy of systems; components, subcomponents, and sub-subcomponents to provide structure to the risk analysis process.

The formal, methodological development of HHM builds on the hierarchical overlapping coordination (HOC) methodology [Macko and Haimes 1978] and the hierarchical multiobjective optimization (HMO) model [Tarvainen and Haimes 1982]. The impact of the HHM methodology is realized in that the basic philosophy is to build a family of hierarchical holographic submodels (HHSs) which address different aspects of the system. With each decomposition represented as its own HHS, HHM provides a construct for consideration of multiple objectives, multiple decision makers, linear and nonlinear causal relationships between model elements, and the coordination between the HHSs. Figure 2.1 depicts a HHM framework with three HHSs and subsequent sub-HHS levels.

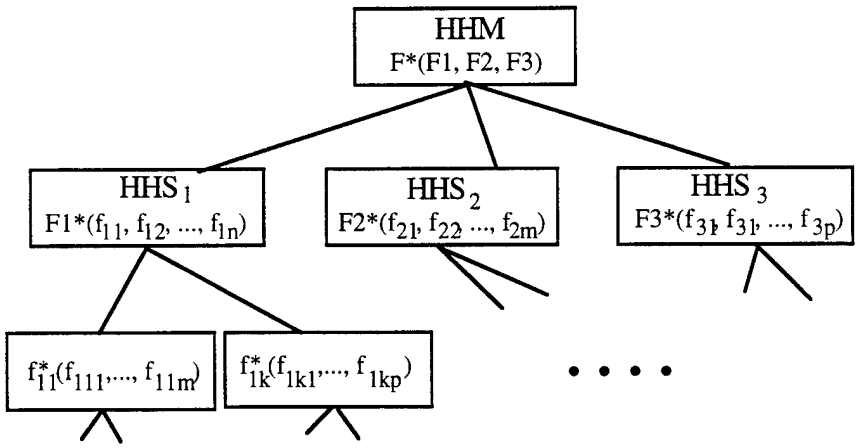


Figure 2.1 Coordination in an HHM framework

2.3.1 HHM as a Model Management Methodology

While not generally referred to as such, HHM can be considered a model management methodology, related to the graphical approach of Muhanna and Pick [1994]. HHM provides a graphical framework for understanding and analyzing the elements (components) of a complex situation, the interrelations of these elements, data requirements, and analytical modeling approaches. In addition, the HHM methodology also provides a mathematical construct for higher-level decision making by resolving the

coordinated solution at the submodel level (and even at sub-submodel and lower levels as necessary), allowing for the possibility of overlapping and coordinated decision variables, objectives, and program constraints -- qualities that are not part of other model management methods.

HHM has been successfully deployed for examining large-scale technical processes and decision problems (e.g., [Haimes et al. 1994b], [Chittister and Haimes 1993], [Haimes et al. 1990]). While there are many complicated dimensions of the software acquisition process, a most troubling issue for software program managers is estimating software cost, schedule, and performance and making the difficult trade-off decisions among these elements [Barrow et al. 1993]. We next review analytic modeling approaches for software estimation.

2.4 Software Estimation

Software estimation is the activity of determining a software project's resource requirements, generally in terms of a project's cost and schedule. Due to the inherent relationship between cost and schedule, most software estimation models provide projections of both elements.

Estimating a software project's cost and schedule is most often assessed by first answering the question [Conte et al. 1986], "how many people will the project require (or person-equivalent units of effort)?" Thus, the result of software estimation is often in terms of *effort*, as opposed to actual cost. The effort estimate can then be converted to cost and schedule.

Software estimation models rely on a combination of algorithmic techniques, expert judgment, and analogy to past data. The algorithmic approach has been the most-studied method, identifying factors that must be considered while estimating development cost and schedule [Boehm and Papaccio 1986], [Mohanty 1981], [Boehm 1981], [Benbasat and Vessey 1980]. Determining the relationships among the factors has been approached through statistical models that rely on analysis of past software project data. The two most widely-applied approaches for software estimation use different measures of system complexity as the basis of their estimation [Barrow et al. 1993]: the thousands of lines of code (KLOC) approach and the function point (FP) approach.

2.4.1 KLOC-based Software Estimation

The largest class of software estimation models relies on mathematical relationships that compute software project development effort and schedule as a function of project size. The number of lines of code is the most commonly-used measure for software estimation. [Blum 1992]. The reliance on KLOC as the principal estimation factor was selected early by researchers, and is based on the observed correlation between delivered lines of code and development effort (measured in man-months) in the data collected from hundreds of software projects (e.g., [Boehm 1981], [Freiman and Park 1979]).

One difficulty with the KLOC-based approach has been establishing a set definition for a "line of code." Discussion in the literature concerning comment lines, declaration statements, etc. has led to a general, although not universal, agreement [Boehm 1981], [Boehm 1986], [Jones 1986].

The relationships contained in KLOC-based models express development effort as a function of the number of lines of code, with the most-common form

$$MM = a + b(KLOC)^c \tag{2.1}$$

where MM is the man-months of development effort required to complete the project . The models focus on producing families of $\langle a, b, c \rangle$ values to account for project-specific factors. Improvements on the basic model (2.1) include adjustment multipliers that reflect project complexity, personnel experience levels, and management control policies (particularly resource allocation strategies).

A conversion equation is then used to translate the man-month development effort estimate into an estimate of the project's development time, t_D

$$t_D = d(MM)^e. \tag{2.2}$$

The values of the parameter vector <*d, e*> depend on project-specific attributes and characteristics. Note that t_D is the development time -- the time requirement *after* the plans and requirements phases are completed.

The projected effort and time duration for each phase of the development life cycle is determined by distributing the total development effort and development time according to past observations of phase-specific effort and time requirements. This is often done on a simple percentage basis [Boehm 1981], or by more complex models that consider manpower staffing rates and other variables such as the trapezoidal staffing model [Londeix 1987] and the Rayleigh curve model [Norden 1966].

2.4.2 Function Point-based Software Estimation

In function point analysis, software estimation is based on the intended system's functional characteristics rather than its predicted size [Albrecht and Gaffney 1983]. Advocates of the approach claim that the method computes the cost of the *problem to be solved* rather than the *product to be delivered.* The objective is to quantify the size and complexity of a software system in terms of the functions that the system delivers to the user.

Function point counts are arrived at by considering a linear combination of five basic software component estimates: inputs, outputs, inquiries, files, and external interfaces. Each component is evaluated at three complexity levels: low, average, and high. An adjustment factor that considers numerous aspects of a software project's processing and development complexity is then employed to modify the base function point count.

Models that use function points for software estimation have generally been approached through regression methods that relate FP to development effort, or FP to development time [Albrecht and Gaffney 1983], [Kemerer 1987], and [Matson et al. 1994]. A limitation to its usefulness, function point analysis is designed to measure business-type applications; it is not appropriate for technical or complex applications that deal with complex algorithms.

Currently, most users of the FP technique employ the method as a way of improving their KLOC estimate within a traditional cost estimation model [Austin and Paulish 1993]. Studies have produced FP-to-KLOC conversion multipliers for several software languages and development environments [Albrecht and Gaffney 1983] [Jones 1986]. Thus, function points normally are employed as an improved method of predicting the size of the delivered product, which is then used to predict effort and schedule.

2.4.3 Software Estimation Models

The two system complexity measures, KLOC and FP, have been employed in a wide range of models. A taxonomy of software estimation approaches is shown in Figure 2.2.

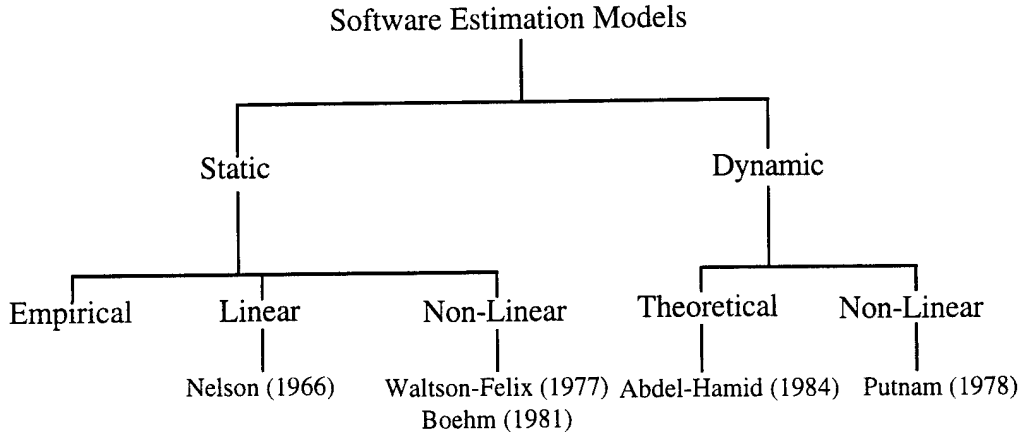


Figure 2.2 A Taxonomy of Software Estimation Models

Studies have shown no single method to be better than all others from all aspects [Bell 1995] (see also [Conte et al. 1986], [Ferens, 1984], and [Kemerer 1987]). The selection of an appropriate estimation model is, therefore, dependent on the attributes of the particular software project, the development environment, software estimation experience, and the availability and type of estimation data (expert and otherwise) [Navlakha 1990].

The majority of software estimation models are of the static class, with very little work having been done in developing dynamical models. Early attempts projected linear relations between software effort and system size [Nelson 1966]. Later models explored more-realistic non-linear relationships. A number of non-linear static models have been proposed, of which one of the first was by Waltson and Felix [1977]. For surveys of linear and non-linear software estimation models, see [Barrow 1993], [Londeix 1987], [Conte et al. 1986], [Boehm 1981], and [Herd 1977]. The most widely known and widely used software estimation model is the COnstructive COst MOdel (COCOMO) developed by Boehm [1981]. Numerous variations of the COCOMO have been developed, with recent updates including consideration of Ada projects [Boehm and Royce 1987], the use of commercial off-the-shelf (COTS) packages, software reuse, and other modern software development process issues [Boehm et al. 1995].

Of the multitude of software estimation models besides COCOMO and its derivatives, a few have gained at least a degree of wide-spread acceptance, including: Price-S [Freiman and Park 1979], SEER-SEM [Galorath 1989], REVIC [Kyle 1991], Checkpoint [Barrow et al. 1993], and SLIM [Putnam 1978]. Of these, SLIM is the only one that could be considered a dynamic model. [Note: The Abdel-Hamid [1984] dynamic model, referred to as a *holistic* representation model [Bell 1995], is a conceptual model that attempts to describe the multitude of dynamic relations in software development with graphical node and arc relations.] Some of these models are KLOC or FP-based, while others rely on different metrics. Unlike COCOMO, whose methodology has been widely published, most of these other models are proprietary and their methodologies not released to the public.

2.4.3.1 COCOMO. While this section provides only a brief review of COCOMO, a detailed tutorial of COCOMO is given in Appendix A. Originally developed in the early 1980s [Boehm 1981], COCOMO is widely recognized within the software community as the predominant software estimation methodology. COCOMO is a public model, in that its methodology, assumptions, projects database, and updates have been widely published (e.g., [Boehm 1981], [Boehm and Royce 1989], [Boehm 1995]). COCOMO consists of three models of increasing complexity: Basic COCOMO, Intermediate COCOMO, and Detailed COCOMO. The primary distinction among the models is the detail and number of model parameters.

Of the three models, the Intermediate COCOMO model has been the most widely implemented and, therefore, the most widely implemented of all software estimation models. The Intermediate COCOMO uses development effort equations to estimate the total man-months (MM) of development effort required to complete a project:

$$MM = (EAF)[a(KLOC)^b] \tag{2.3}$$

where the parameter vector $<a,b>$ takes on differing values according to the development mode of the project, and the effort adjustment factor (EAF) indicates the effect of 15 "cost driver" attributes. The value MM that is produced from the effort equation is used in the schedule equation to estimate development time t_D (in months)

$$t_D = c(MM)^d \tag{2.4}$$

where the parameter vector $<c,d>$ takes on differing values according to the development mode of the project.

Recent extensions of the original COCOMO include a version tailored for Ada language projects [Boehm and Royce 1987], [Boehm and Royce 1989]. Development of a COCOMO model that includes consideration of software reuse and re-engineering, commercial off-the-shelf (COTS) packages, object orientation, non-sequential process models, and rapid development processes is in the design stage [Boehm et al. 1995].

In accuracy tests using actual KLOC and effort multiplier data, the reported accuracy of the COCOMO models is reported to be (at best) within 20% of the actual results 68% of the time [Barrow et al. 1993]. These reported accuracy levels define the upper limit of accuracy for software cost estimation, as these have been evaluated using actual, ex-post data. The COCOMO models are often used as the benchmark for testing other estimation methodologies [Kemerer 1987] and serve as the standard for the software community.

2.4.3.2 The Price-S Model. The Price-S model was developed by GE Price systems primarily for aerospace applications [Freiman and Park 1979] [Wolverton 1980]. A proprietary model, the exact form of the model's equations is not readily available. A general description of the model's methodology, however, is found in [Price 1988]. The major input to Price-S is KLOC; other inputs include software functions, operating environment, complexity factors, and productivity factors. As the model is intended for a specific software domain, its use for business and other non-aerospace applications is questionable.

2.4.3.3 SEER-SEM. The System Evaluation and Estimation of Resources - Software Estimation Model (SEER-SEM) provides software estimation through knowledge bases developed from completed projects [Galorath 1989]. While the actual algorithms of the model are proprietary, SEER-SEM permits either KLOC or FP sizing input, and has been updated to handle software reuse and COTS development projects [McRitchie 1995]. SEER-SEM is applicable to all types of software projects and considers all phases of software development [Stutzke 1995].

2.4.3.4 REVIC. The Revised Enhanced Version of Intermediate COCOMO (REVIC) was developed by Hughes Aerospace using a database of DoD aerospace software projects [Kyle 1991]. Although REVIC is a COCOMO derivative, it uses different coefficients in the effort equations and uses a different methodology for distributing effort and schedule to each phase of product development.

REVIC also applies a measure of standard deviation to each estimate as a means of risk assessment. REVIC provides a single-weighted "average" distribution for effort and schedule along with the option for the user to vary the percentages in the development phases.

2.4.3.5 Checkpoint. The Checkpoint model is a knowledge-based software estimation model with algorithms derived from measurements of more than 4200 software projects [Barrow et al. 1993]. The model is based on the work of Jones [1986] and incorporates proprietary algorithms. Checkpoint was one of the first models to incorporate function points as a measure of size to estimate project complexity. Other model inputs include project type and class, experience level, development method and environment. The model is intended to be applicable to all types of programs and all phases of the software development life cycle.

2.4.3.6 SLIM. While COCOMO and related models may be considered a micro-modeling approach (evidenced through reliance on detailed modeling parameters and an extensive projects data base), the Software Life Cycle Model (SLIM) could be considered a macro model that offers a top down approach to software estimation [Londeix 1987]. Developed by Putnam [1978] [Putnam and Fitzsimmons 1979], the SLIM model is based on Putnam's analysis of the software life cycle in terms of the Rayleigh distribution of project personnel level versus time (hence the dynamical model classification).

While SLIM is a proprietary model and much of the detail regarding its current form is not publicly available, the general theory upon which the model is based is available in the open literature [Putnam and Myers 1992]. Originally developed from analysis of ground-based radar software programs, SLIM has been expended to include other types of programs [QSM 1987].

The SLIM model considers four essential variables that characterize a software project. In addition to estimating the size of the project (in KLOC), the user must estimate two of the remaining three parameters: life cycle manpower requirement (in man-months) indicating the software developer's staffing throughout the life cycle, the productivity factor (PF) that measures the software developer's efficiency, and the total required development time t_D. The relationship among model variables is given by [Putnam and Myers 1992]

$$KLOC = (PF)\,MM^{1/3} t_D^{1/3} \tag{2.5}$$

which, by rearranging the equation, can be solved for the unknown variable.

The Rayleigh curve orientation of SLIM provides a framework for investigating effort-schedule trade-offs, but experience has shown that the SLIM model is very difficult to implement without considerable experience and extensive project-specific tailoring [Putnam and Myers 1992].

2.4.4 Accuracy of Software Estimation Models

Objective studies of software estimation methods have been few. Often, the developers of a method have described their own technique and reported their own assessment [Jensen 1983], [Putnam 1978], [Waltson and Felix 1977]. Other researchers have tried to predict the cost of software projects but only after their completion and hence with full knowledge of their final scope [Banker and Kemerer 1989], [Kemerer 1987], [Kitchenham and Taylor 1985]. Studies found the error rates of cost estimation models ranged from 85% to 772% [Kemerer 1987], while comparative studies of expert estimation (without algorithmic models) showed error rates ranged from 32% to 1107% [Vicinanza et al. 1991].

Regardless of the underlying independent variables, studies indicated the tight link between the preparation of the algorithm's input parameter estimates and the accuracy of the model's final prediction [Kemerer 1987], [Lederer and Prasad 1993]. In terms of the two principal approaches, studies indicate estimating KLOC (versus estimating the many parameters of the FP approach) proved to be a more manageable task [Lederer and Prasad 1993], [Bailey 1986]

2.4.5 Software Estimation Tools

There are many software tools on the market that implement the software estimation models described above. A list of software cost estimation tools is provided in Table 2.1, grouped according to the underlying model or methodology employed: COCOMO, Function Point, or other. Other than those tools that are direct implementations of COCOMO or FP, the majority of these tools contain proprietary algorithms and datasets. A brief description of each tool, along with information concerning its vendor or supporting organization is provided in Appendix B.

Table 2.1 Software Estimation Tools

Product	Comments
COCOMO-Based Tools	
CB COCOMO	Based on COCOMO to estimate effort and cost of software development projects
COCOMO1	Artificial Intelligence front end with COCOMO model
COCOMOID	Provides estimates also based on enhanced ADA, Ada process and incremental development models.
CoCoPro	Estimates resources needed using standard COCOMO
COSTAR	Uses Detailed COCOMO, Ada COCOMO, and allows sizing with function points.
COSTMODL	Detailed COCOMO model; available to government by NASA/JSC.
GECOMO Plus	Implements an extended Detailed COCOMO and includes Ada.
GHL COCOMO	Estimates development cost based on COCOMO.
REVIC	Intermediate COCOMO with life cycle costing and risk analysis (public domain)
SECOMO	Full COCOMO with maintenance cost estimation.
SWAN	Cost estimation with COCOMO, size estimates with function point analysis. Developed for US. Army.
Function Point-Based Tools	
ASSET-R	One in a family of models for software development estimation. Uses FP analysis.
CA-FPXpert	On-line tutor for FP analysis.
CHECKPOINT	Knowledge-based estimation tool using FP.
Micro Man ESTI-MATE	Estimates for Information systems using FP analysis.
PROJECT BRIDGE	Knowledge-based tool using FP analysis for Information Systems projects.
SIZE Plus	FP analysis for data processing and real-time applications.
SPQR/20	Incorporates proprietary algorithms with FP analysis for cost and productivity estimation.
Other Method-Based Tools	
CA-ESTIMACS	Estimates effort, schedule, and cost of Information System projects.
CEIS	Four independent size estimates based on comparison to known projects.
COSTEXPERT	Expert System based model. Does not use KLOC or FP.
PRICE S	Uses functionality and KLOC for cost, effort, and schedule estimates.
SASET	Forward chaining, rule-based expert system utilizing a hierarchically-structured knowledge database.
SEER-SEM	Software cost, schedule, and risk estimation model (Air Force-wide license).
SEER-SSM	Software size estimator.
SIZE PLANNER	Uses four approaches for size estimation: fuzzy logic, FP, standard component, and new/reused/modified sizing.
SIZEEXPERT	Produces estimate of KLOC based on cost expert questions.
SLIM	Proprietary analytic tools and expert system methodology for cost and schedule estimation.
SOFTCOST-R	KLOC-based model for estimation of general projects.
SYSTEM-4	Cost estimation utilizing KLOC-based proprietary algorithm.

2.5 Software Performance

Software performance generally refers to how well the software *functions* to meet the stated requirements of the system [Musa et al. 1990]. Software performance includes many attributes such as availability, reliability, safety, performability, maintainability, and testability [Johnson 1989]. Reliability is probably the most important of the characteristics [Bittanti et al. 1988]. It is intimately connected with defects, and as Jones [1986] points out, defects represent the largest cost element in programming. In this section we define the common terms associated with software reliability and discuss various approaches of modeling and quantifying software reliability. The effect of management policy options on software reliability is discussed.

2.5.1 Software Defects, Faults, Errors, and Failures

We must first distinguish between software faults and software failures. A software *fault* is a *defect* in the program that, when executed under particular conditions, causes an error [Musa et al. 1990], [Johnson 1989]. A software *failure* is the departure of the external results of program operations from requirements [Musa et al. 1990]. Johnson [1989] includes an intermediate term, software *error,* as a manifestation of a fault which then leads to the observed failure. More commonly, the distinction is limited to the two: faults and failures [Putnam and Myers 1992], [Musa et al. 1990], [Bittanti et al. 1988].

Software faults (we'll also refer to these as defects) occur throughout the development process and may occur in requirements, specifications, or design, as well as in the actual computer code [Putnam and Myers 1992]. Software developers find these defects by means of self-checking, reviews, walkthroughs, inspections, module testing, etc., not simply because the program fails in operation.

The number of defects and the resulting observed failures are key values for determining software reliability. The software *defect rate* is the number of defects per unit of development time, or the rate at which defects are introduced into the software system over time. A related concept is that of *defect density*, which is the number of defects per size unit of code (e.g., KLOC). The software *failure rate*, as opposed to the defect rate, represents the number of program failures per unit of execution or operation time.

2.5.2 Software Reliability

The definition of software reliability that is widely accepted throughout the field is the probability of failure-free operation for a specified time interval [t_0, t] given the system was performing correctly at time t_0 [Johnson 1989]. Letting T be the random variable representing the time to next failure, the reliability function $R(t)$ is given by

$$R(t) = P[T > t]. \tag{2.6}$$

Associating to the random variable T the cumulative distribution function (cdf) $F(t)$ and the probability density function (pdf) $f(t)$, we observe

$$F(t) = \int_0^t f(x)dx = P\left[T \le t\right] = Q(t) = 1 - R(t). \tag{2.7}$$

$Q(t)$ is the *unreliability* function, and $f(t)$ is the *failure density* function. From Eqs. (2.6) and (2.5), we note that reliability is equivalent to an exceedance function, hence its natural suitability for extreme event analysis [Asbeck and Haimes 1984].

A significant quantity that provides an index of reliability is the mean time to failure (MTTF). The MTTF is the expected time that a system will operate before the first failure occurs. It is defined as [Johnson 1989], [Bittanti et al. 1988]:

$$MTTF = \mathrm{E}\left[T\right] = \int_0^\infty tf(t)dt = \int_0^\infty R(t)dt \tag{2.8}$$

The failure rate function $z(t)$ is the instantaneous rate at which the software fails. The failure rate can be expressed in different ways, most simply

$$z(t) = \frac{f(t)}{R(t)}. \tag{2.9}$$

Software reliability information is valuable for a number of reasons [Austin and Paulish 1993]:

- As the failure rates decrease with testing and debugging, predictions of failure rates can help determine when to stop testing (e.g., when the quality of the software is adequate).
- Having a prediction of failure rates makes decisions about trade-offs among performance, cost, schedule, reliability, and other factors easier and more explicit.

- Accurate prediction of failure rates makes it possible to guarantee failure rates below certain tolerances.

2.5.3 Approaches to Highly Reliable Software

Essentially there are three different ways to pursue highly reliable software: to avoid the occurrence of faults in the design and development of the program; to make use of fault tolerant structures; to remove faults during the test phase.

The first approach consists of designing software by using structured programming, formal specification languages, software clean rooms, or other software development tool effective for reducing the probability of error introduction [Lyu and He 1993], [Ghezzi et al. 1988], [Gehani and McGettrick 1986].

In the second approach, reliability is obtained by designing fault-tolerant software systems, able to perform satisfactorily even in the presence of faults [Johnson 1989]. This is usually done by providing convenient redundancy in the program or adding some error-recovery procedures. Surveys and comparisons of fault tolerant software systems and their effectiveness in terms of software reliability can be found in [Hudak et al. 1993], [Vaidya and Pradhan 1993], [Tai et al. 1993], [Kanoun et al. 1993], [Knight and Ammann 1991], and [Avizienis 1985].

The third approach improves software reliability through testing and fault correction activities ("debugging"). During this phase, the program undergoes an extensive validation test aimed at detecting as many remaining defects as possible. As soon as a failure is observed, the fault in the code which caused the failure is searched for and, hopefully, eliminated. Despite advances in the other two areas, thorough testing is the primary method to achieve software reliability and usually takes a significant percentage of time in the life cycle of the software product. Common test-for-reliability issues include estimating the inherent fault density, determining fault reduction factors, fault detectability, fault correctability, the introduction of new faults while correcting others, fault prioritization, and the relationship between testing time and defect detection and removal [Rozum 1992], [Musa et al. 1990], [Bittanti et al. 1988], [Putnam and Myers 1992], [Jones 1986].

2.5.4 Software Reliability Models

To model software reliability, one must consider the principal factors that affect it: fault introduction, fault detection and removal, and the environment. Fault introduction depends on the characteristics of the developed code and the development process characteristics. The most significant code characteristic is size (KLOC) [Musa et al. 1990], [Boehm 1981]. Development process characteristics include software engineering technologies and tools used and the level of experience of the personnel. Fault removal depends on time, operational profile, and the quality of the repair activity. The environment directly depends on the operational profile.

Various analytical methodologies have been applied to the prediction of software reliability. These methodologies differ with regard to the level of data, the nature of the dependent variables, and the development outcomes under consideration. Regression models have been developed from past project data that relate reliability metrics (e.g., defect density) with other system variables such as KLOC, programmer experience, computer language, and development time [Evanco and Lacovara 1994], [Putnam 1992], [Musa et al. 1990], [Agresti and Evanco 1992], [Prentice 1981], [Boehm 1981]. These relationships are then expressed in algorithmic models to estimate software reliability given certain project characteristics.

As some of the above factors are probabilistic in nature and operate over time, other software reliability models are formulated in terms of random processes. The models are distinguished from each other in general terms by the probability distribution of failure times or number of failures experienced and by the nature of the variation of the random process with time. Poisson process models of the nonhomogeneous type have been used to describe the random failure process and estimate system reliability [Schneidewind 1993], [Kanoun et al. 1993], [Musa et al. 1990], [Bittanti et al. 1988].

For software systems that can be represented by modular structures or that incorporate redundant-design architecture, Markov models have been applied to estimate software system reliability [Johnson 1989], [Tai et al. 1993], and [Kanoun et al. 1993]. The Markov modeling framework allows for analysis of fault introduction, error occurrence and detection, and recovery or failure. The system states are generally defined in terms of operating status (e.g., good, error, error-detected, recovered) and completion status (e.g., pass, fail, crash, abort) [Hudak et al. 1993]. Markov models utilize estimated hazard rates,

MTTF data, correction rates, and detection capabilities to define the state transition probabilities.

2.5.5 Software Reliability Trade-offs

While software reliability models provide the behavior of the software failure rate over time, this relationship is also tied to other software development parameters: product size, development time, staff size and effort, productivity, project complexity, acceptable performance thresholds, and manpower buildup rate [Putnam and Myers 1992]. Each of these parameters has an effect on the number of defects created and, consequently, upon the reliability of the product. In effect, more or less of each parameter can be traded-off for reliability. Of these trade-offs, some fall within the domain of the customer and user (acceptable performance levels, product size and complexity - as affected by requirements) while others are under the control of the contractor (staffing, personnel experience, etc.).

2.6 Probabilistic Evaluation

Most, if not all, large-scale systems are designed, managed, maintained, and utilized under the condition of uncertainty. Unfortunately, this uncertainty is often neglected in the formulation of supporting analytic models. Software estimation is no exception.

In most cases, the traditional use of a single value estimate is not adequate to represent the uncertainty associated with that system or environment. Because the assessment and ultimate prevention of risks (e.g., cost overrun) are of such major concern, it is desirable to retain as much information as is possible in the quantification of model parameters. A single value assessment conveys no information as to the likelihood (let alone possibility) of deviation from that assessed value. It is for this reason that it must be advocated that more than a single value assessment be conducted.

Probabilistic quantification is one approach for addressing the problems associated with single value estimation. A probability distribution is a model able to capture the range and estimated likelihoods of the potential realizations of a varying or uncertain quantity. Even in the face of little empirical evidence, probabilistic evaluation can be conducted using the triangular distribution [Haimes et al. 1994a]. Constructing a triangular distribution requires evaluation of only three values: the lower bound, the upper bound, and the most likely

value. Use of the triangular distribution is widespread; the Army Corps of Engineers accepts the triangular distribution as the basis for their probabilistic analysis for flood and river risk management [Haimes et al. 1994b].

2.6.1 Fallacy of the Expected Value

Unfortunately, in most cases where probabilistic quantification is practiced, the expected value is used as the sole measure of risk. This may lead to inaccurate or misrepresentative results. The expected value of risk is an operation that takes the product of each outcome and its probability of occurrence and sums (or integrates for continuous outcomes) all these products. This operation thus commensurates adverse events of high consequences and low probabilities with events of low consequences and high probabilities. The expected value thus masks the true variance and uncertainty that should be represented. This phenomena is illustrated in Figure 2.3. Notice that both events A and B have the same expected value but have very different distribution characteristics .

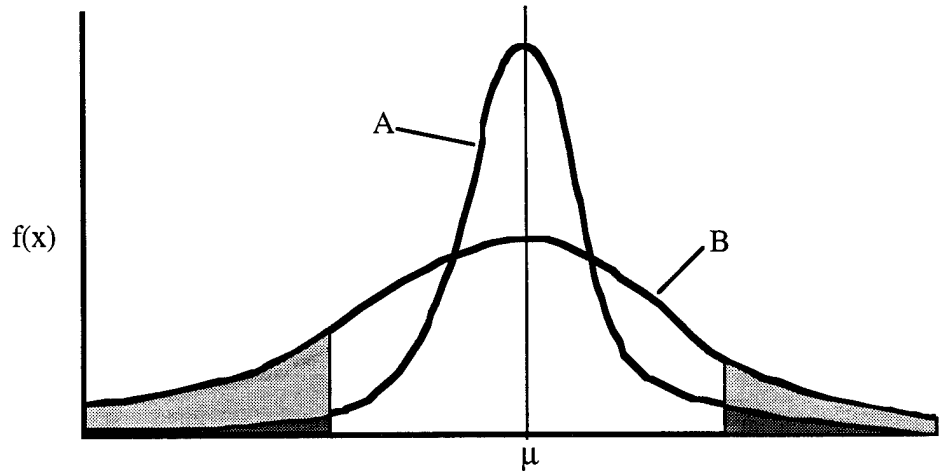


Figure 2.3 Insufficiency of the expected value for decision making

Distribution B has greater variance, or a greater tendency for events to occur in the extreme ranges of the distribution. Given only the expected value of the two events, a decision maker would be indifferent between the two events, but given the "full" picture of the two events the decision maker would choose A, the event with a lower risk of extreme events. This simple example provides a strong argument for the need to consider more than the expected value as the principal metric used in decision making.

2.6.2 The Partitioned Multiobjective Risk Method (PMRM) and the Conditional Expected Value

Catastrophic, or extreme events, are defined as having a low probability of occurrence but a high damage level. Proper risk management must focus on the management of extreme events -- for it is these catastrophic disasters, not the more common "expected value" events, that cause grave harm to the system. A risk measure associated with extreme events -- the conditional expectation -- can be useful in management of extreme events as it does not average out catastrophic events with more high-frequency, low-consequence events.

Consider a continuous random variable X of damage (e.g., cost overrun, time delay) that has a cdf $F(x)$ and a pdf $f(x)$, which are defined by the relationships

$$F(x) = \mathrm{P}[X \le x], \quad x \ge 0 \tag{2.10}$$

and

$$f(x) = \frac{dF(x)}{dx}, \quad x \ge 0. \tag{2.11}$$

The cdf represents the *nonexceedance probability* of x, the probability that X is observed to be less than or equal to some value x. The *exceedance probability* of x is defined as the probability that X is observed to be greater than or equal to x, and is equal to one minus the cdf evaluated at x. The expected value, average, or mean value of the continuous random variable X is defined by

$$\mathrm{E}[X] = \int_{-\infty}^{\infty} x f(x) dx. \tag{2.12}$$

In the partitioned multiobjective risk method (PMRM) [Asbeck and Haimes 1984], the concept of the expected value is extended to generate the *conditional expected-value function*, often referred to as a risk function, which is associated with an extreme range of exceedance probabilities or their corresponding range of extreme harm severity. The resulting conditional expected-value function, in conjunction with the traditional expected value, provides a new measure of the risk of extreme events associated with a particular policy.

Referring to Figure 2.4, let $(1-\alpha)$, where $0 \le \alpha \le 1$, denote an exceedance probability that partitions the domain of X into a range of extreme events, as follows. On a plot of exceedance probability, there is a unique harm β on the damage axis that corresponds to

the exceedance probability $(1-\alpha)$ on the probability axis. Damages greater than β are of high severity and low exceedance probability and constitute the range of extreme events. If, for example, $(1-\alpha)$ is taken to be equal to 0.05, then β is the 95th percentile.

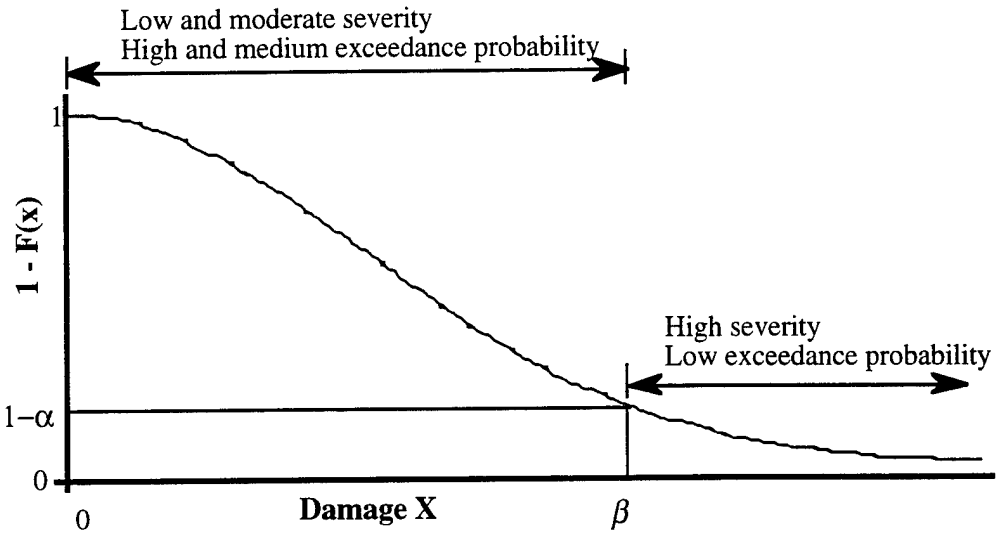


Figure 2.4 Extreme event probability partitioning

For a range of extreme events, the conditional expected damage (given that the damage is within that particular range) provides a measure of the impact associated with extremely large and potentially very costly and catastrophic events or scenarios. This measure is based on the definition of the *conditional expected value*. The conditional expected value risk measure is denoted by f_4 and is related to scenarios of low exceedance probability and high severity. The function f_4 is the expected value of X, given that x is greater than β:

$$f_4 = \mathrm{E}[\mathrm{X} \mid x \geq \beta] = \frac{\int_{\beta}^{\infty} xf(x)dx}{\int_{\beta}^{\infty} f(x)dx} \tag{2.13}$$

Thus, for a particular policy option, there is the additional measure of risk f_4, in addition to the traditional, unconditional expected value, E[X], denoted by f_5. Note the similarity between the forms of f_4 and f_5:

$$f_5 = \frac{\int_{-\infty}^{\infty} xf(x)dx}{\int_{-\infty}^{\infty} f(x)dx} = \int_{-\infty}^{\infty} xf(x)dx = \mathrm{E}[\mathrm{X}] \tag{2.14}$$

(since the probability in the denominator of Eq. (2.14) is necessarily equal to one). In the PMRM, the conditional and unconditional expected values are balanced in a multiobjective formulation. The function f_4 is related to the percentile (e.g., the 95th percentile) as a measure of the risk of the extreme events, but provides a superior representation of the risk of extreme events because it can distinguish between different shaped distributions when the exceedance probability at a particular location is the same. The function f_4 is a natural measure of the risk of extreme events for risk-based decision making. It transfers the positive attributes of the unconditional expected value to a preselected extreme range of harm, thus providing a basis for realistic and conservative policies.

2.7 Chapter Summary

This chapter has reviewed topics pertinent to software acquisition risk management. The complexities of the software acquisition process, in particuar the numerous elements asssociated with the software estimation process and software reliability require a multi-visionary approach to software acquisition management. The holistic framework of HHM provides the basis for systematic investigation of the many dimensions of software acquisition. Probabilistic extensions of software estimation models will explicitly incorporate the uncertainties associated with the software estimation effort. Through the HHM, appropriate software estimation models can be brought together, allowing for coordinated trade-off analysis and risk-based decision making that utilizes the additional information of the conditional expected value of extreme events.

Chapter 3
A Holistic Management Framework for Software Acquisition

This Chapter develops a holistic framework for software acquisition that provides a comprehensive representation of the multitude of elements, issues, activities, organizations, and products that taken together constitute software acquisition. Centered on hierarchical holographic modeling (HHM), the framework has provisions for exploration and expansion of the multiple visions or decompositions. The intent of this Chapter is not to explicitly address and consider all of the multiple aspects associated with the software acquisition process; rather, the objective is to develop a framework that would enable the consideration of such complexities and interconnectedness. This framework forms the foundation of a systemic approach to software risk identification, software cost and schedule estimation, and software project management decision making -- topics that are addressed in subsequent chapters of this work.

3.1 Introduction

Effective management of modern, complex processes such as software acquisition requires capable, mature direction. To do justice to the management of technological systems, one must address the holistic nature of the system in terms of its hierarchical, organizational, and functional decision-making structure; the various time horizons; the multiple decision makers, stakeholders, and users of the system; and the host of technical, institutional, legal, and other socioeconomic conditions that require consideration. With the ever-increasing importance and complexity of the software component of modern systems, it is essential that software acquisition be addressed in terms of its overall system.

The role of models is to represent the intrinsic and indispensable properties that serve to characterize the system, i.e., good models must capture the essence of the system. "In the abstract, a mathematical model may be viewed as a one-sided limited image of the real system that it portrays. To clarify and document not only the multitude of components, objectives, and constraints of a system but also its welter of functional, temporal, and other aspects is quite impossible with single-model analysis and interpretation" [Chittister and Haimes 1993]. Given this assumption and the notion that ever-present integrated models

cannot adequately cover a system's aspects per se, the concept of HHM constitutes a comprehensive theoretical framework for systems modeling.

Clearly, the multi-dimensionality of the acquisition process, and the large number of activities, organizations, and disciplines that are engaged in this process defy the capability of any single model to represent the essence of the acquisition process. To overcome the shortfalls of single planar models and to identify all sources of risk associated with the software acquisition process, an HHM framework will be adopted here. HHM assumes an iterative approach to providing the structure for identifying all risks. If one fails to identify a risk source with the current views of the HHM, then expansion of the model to include a new decomposition is possible. This process, itself, will eventually capture all risk sources.

In the remaining sections of this Chapter, the five major decompositions (visions) of the software acquisition HHM framework are described first, followed by a discussion of their integration within the overall HHM structure for risk identification and model development.

3.2 The HHM Decompositions for Software Acquisition

Development of an HHM model for software acquisition requires consideration of the multitude of issues and factors associated with the process. As the intent of an HHM model is to identify and manage risk sources, the elements of the model should be as comprehensive as possible. Specifically, we seek the set of decompositions or visions that together describe the multi-faceted nature of the problem. The materials presented in Chapters 1 and 2 regarding the complex issues affecting software acquisition assist in formulating the dimensions of the problem: the stages and activities of the process, the participant communities, consequences of mismanagement, and risk causation and sources. Investigations into the software acquisition field (e.g., [Sage 1995], [Chittister and Haimes 1995], [Chittister and Haimes 1993], [Blum 1992]) underscore the criticality of a multi-vision approach: the need to consider a continuously changing environment, the pace of technology advancement, and organizational considerations. The Software Development Risk Taxonomy [Carr et al. 1993] provides additional insight regarding the range of issues and concerns affecting the software product in terms of product engineering, development environment, and program constraints.

The HHM for software acquisition developed in this Chapter includes five principal decompositions, perspectives, visions, or hierarchical holographic submodels (HHSs) (see Figure 3.1). Each HHS addresses software acquisition from one particular perspective or dimension. In their totality, these seemingly disparate visions of sources of risk constitute the building stones of a risk identification framework. The *program consequence* HHS represents those outcomes or effects that provide a measure of acceptability for project progress and program management policies. The *community maturity* HHS recognizes the competing, yet coordinated activities, interests, objectives, and concerns of each participant group. The *life cycle* HHS contributes a temporal perspective, accounting for the progression of events and activities of the software life cycle. The *modality* HHS represents the major classes of failures, and the interaction of these failures, that contribute to system risk. The *project elements* HHS considers the project complexity and the development environment.

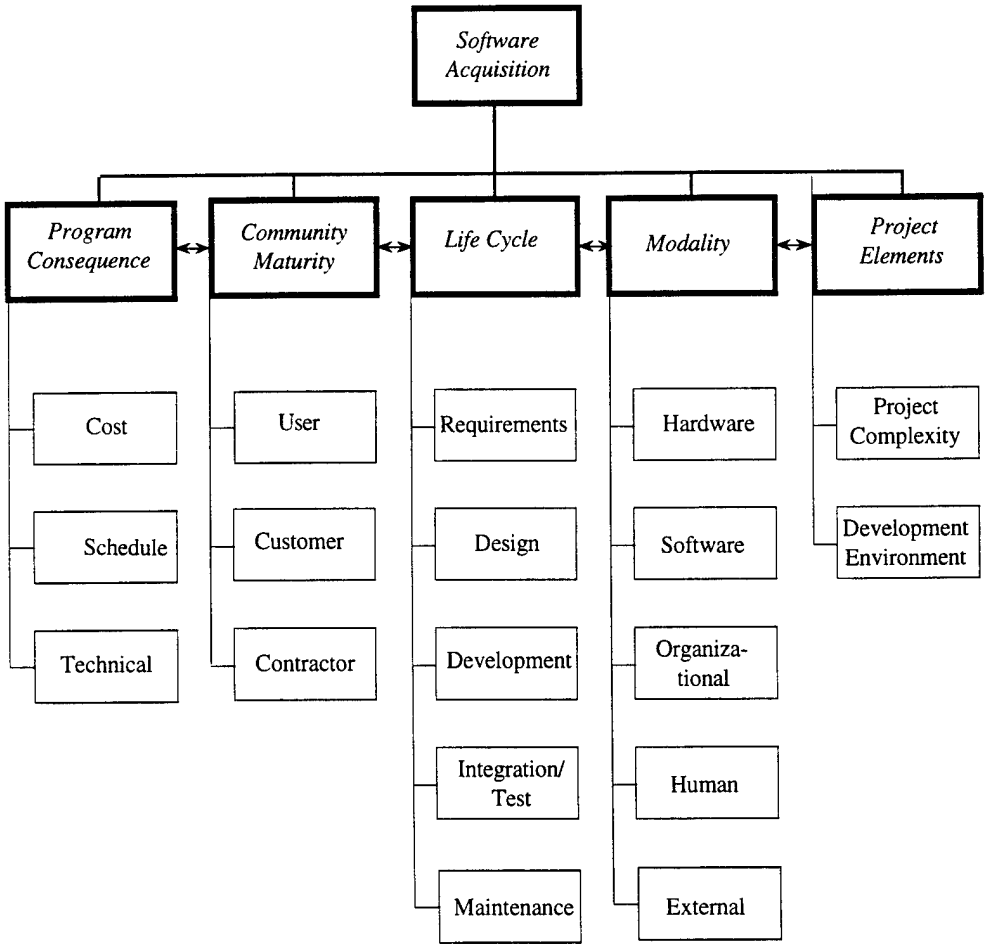


Figure 3.1 Hierarchical holographic modeling for software acquisition

The claim is not made that the HHM in Figure 3.1 is all-inclusive or is a totally comprehensive set of software acquisition risk sources. It does, however, provide the framework that, as our knowledge of software acquisition improves over time, new aspects or visions that are not foreseen today can be added.

3.2.1 Program Consequences Decomposition

One of the visions or decompositions that constitutes the HHM for the risk assessment and management of software acquisition is the perspective of program consequences. This decomposition represents the principal elements that describe the acceptability and progress of an acquisition program: cost, schedule, and technical performance.

1) *Cost*: The programmatic costs associated with software acquisition. Program costs are most often measured against the planned or budgeted expenditures, and costs in excess of these amounts signals potential problems.

2) *Schedule*: The time requirement to design, develop, test, and implement the software system. The total schedule requirement is often segmented into the several life cycle phases for more accurate project tracking.

3) *Technical*: The ability of the system to meet its intended functions and performance requirements.

This vision incorporates the notion of quality, addressing both technical and nontechnical risks: technical performance of the product, cost overrun, and time delay in schedule. To address the risk associated with the program consequences, one must address the overlapping with all other visions and their sub-elements.

3.2.2 Community Maturity Decomposition

Another vision of the HHM can be obtained through consideration of the three principal participant communities associated with a software acquisition endeavor, as described in Chapter 1: the user, customer, and contractor communities.

1) *User*: The community responsible for identifying operational needs (hence, system requirements) and for using and operating the developed system.

2) *Customer*: The organization that works with both of the other communities to procure a system that meets the operational requirements, and do so within budgetary and time constraints.

3) *Contractor*: The organization responsible for developing the system that will satisfy the stated requirements.

The participant community decomposition provides an investigative framework for examining the risk sources within and among each group, relative to the other decompositions. The capabilities, maturity, experience, and competence of each community in performing their required functions and activities is addressed through this vision.

3.2.3 Life Cycle Decomposition

The life cycle decomposition represents the temporal element of software acquisition; the progression of events and activities, along with the sequence of associated decisions, through the various phases of the life cycle. These phases, which may be represented in a waterfall, spiral, or other process paradigm are described in Chapter 1 and include:

1) *Requirements*
2) *Design*
3) *Development*
4) *Integration/Test*
5) *Maintenance.*

These categories provide a comprehensive scheme for the identification and assessment of the sources of risk associated with each category at different points in time, and with all the other decompositions (within the other four visions of the HHM). This decomposition also addresses the propagation of the development effort within the life cycle of a product. For example, the impact of requirements changes over time.

3.2.4 Modality Decomposition

The modality vision addresses five general sources of possible risk of failures:

1) *Hardware*: the hardware used to develop and test the software, as well as the hardware components with which the software is to be integrated.

2) *Software*: includes consideration of various software development technologies (COTS, re-use, etc.) as well as software languages and software used to develop other software (e.g., CASE).

3) *Organizational*: the institutional procedures, regulations, policies, structures, philosophies, maturity, and influence that contribute to program risk.

4) *Human*: software development and acquisition is, fundamentally, a human endeavor - thus is affected by human failures and shortcomings.

5) *External*: this element is to provide coverage of sources of risk that are essentially beyond the control of the software acquisition domain. Examples of external risk sources include political influence, international events, etc.

3.2.5 Project Elements Decomposition

The project elements decomposition includes two sub-decomposition:

1) *Development environment*: consideration of the environment within which the software is to be developed: the software process maturity, software development tools and systems, the work environment, etc.

2) *Project complexity*: the relative degree of difficulty and effort that is to be expected in developing this product. Considers project size, functions, platform, etc.

This decomposition contributes an understanding of the overall complexity of the intended system and the risks that are associated with that complexity. It also captures the sources of risk due to the development environment.

3.2.6 Adding Detail to the HHM Decompositions

In many ways, Figure 3.1 does not do justice to the task of communicating the myriad of risk sources associated with each decomposition element. As mentioned above, however, the HHM provides the framework for expanded analysis and investigation. Consider, for example, the two sub-decompositions of the *project elements* decomposition. Additional analysis provides sub-elements for these sub-decompositions: project complexity includes: size, re-use, functions, and platform; development environment includes: process maturity, tools, systems, and work site (Figure 3.2).

Similar exploration of the other decompositions and their sub-decompositions leads to the inclusion of additional elements within the HHM, providing increased understanding and comprehension concerning software acquisition and the risks associated with each element.

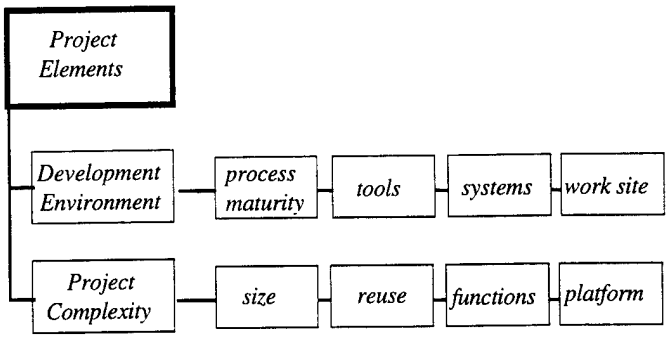


Figure 3.2 Expanding the detail of the *program elements* decomposition

3.3 HHM for Software Acquisition Risk Identification

The HHM model of Figure 3.1 does not fully represent the HHM concept to the reader. The most critical shortcoming of Figure 3.1 is that the HHM philosophy builds on a multidimensional representation of the system (in this case, the multidimensional representation of the sources of risk in software acquisition). As Chittister and Haimes note [1995], "the two-dimensional depiction of the HHM [as in Figure 3.1] conceals the couplings, interconnectedness, and the interactions among the various subsystems that constitute the sources of risk." This said, a systemic exploration of software acquisition risk source relations and interactions can be conducted using the HHM model. Each of the five decompositions can be viewed as the primary vision from which to assess program risks. The HHM framework then enables one to trace, assess, and analyze all other factors affecting and affected by these primary sources of risk. Figure 3.3 depicts one such representation from the perspective of the *program consequence* decomposition, focusing on identifying the cost, schedule, and technical risks associated with each participant community.

Continued investigation, by systematically selecting each decomposition as the primary vision and associating it with the other decompositions, will provide a comprehensive coverage of the sources of risk associated with software acquisition.

3.4 HHM for Analytic Model Development

Systemic investigation of software acquisition risk sources, as described in the previous section, provides increased understanding of the relationships, dependencies, causation, and impacts of decomposition elements on each other. Such understanding permits the

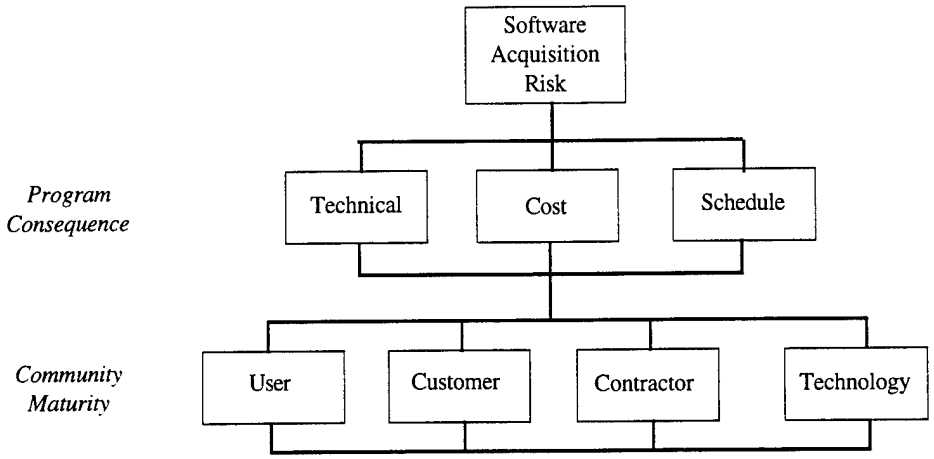


Figure 3.3 Risk assessment - program consequences-based HHM structure

HHM decomposition elements to be used in the development of analytic models. Representing the knowledge of decomposition element relationships in an influence diagram-type structure provides the next level of detail required for an analytic modeling effort.

Consider, for example, the model depicted in Figure 3.4. Investigation using the HHM reveals a relationship between the *program consequence* and *project element* decompositions. Furthermore, *project complexity* is found to be influenced by the intended *size* of the system, its *functions*, opportunity for software *re-use*, not to mention the system *requirements*. In addition, the capabilities and experience of project *personnel* (e.g., analysts and programmers) and the selected *technologies* also affect project *cost*, *schedule*, and *technical performance*. Desired system requirements are identified by the user, with the aauthorized system requirements approved by the customer community. Selection of personnel and technologies for the actual development effort are generally within the contractor's decision domain. With even such simple investigation of element relationships, a framework of an analytic model for software cost, schedule, and technical performance takes form. This graphical element-relationship extension of HHM approaches the graphical model management concept of Muhanna and Pick [1994]. A series of interrelated analytic models can be now developed, with the selection of particular areas for modeling based on the interests and needs of the decision makers.

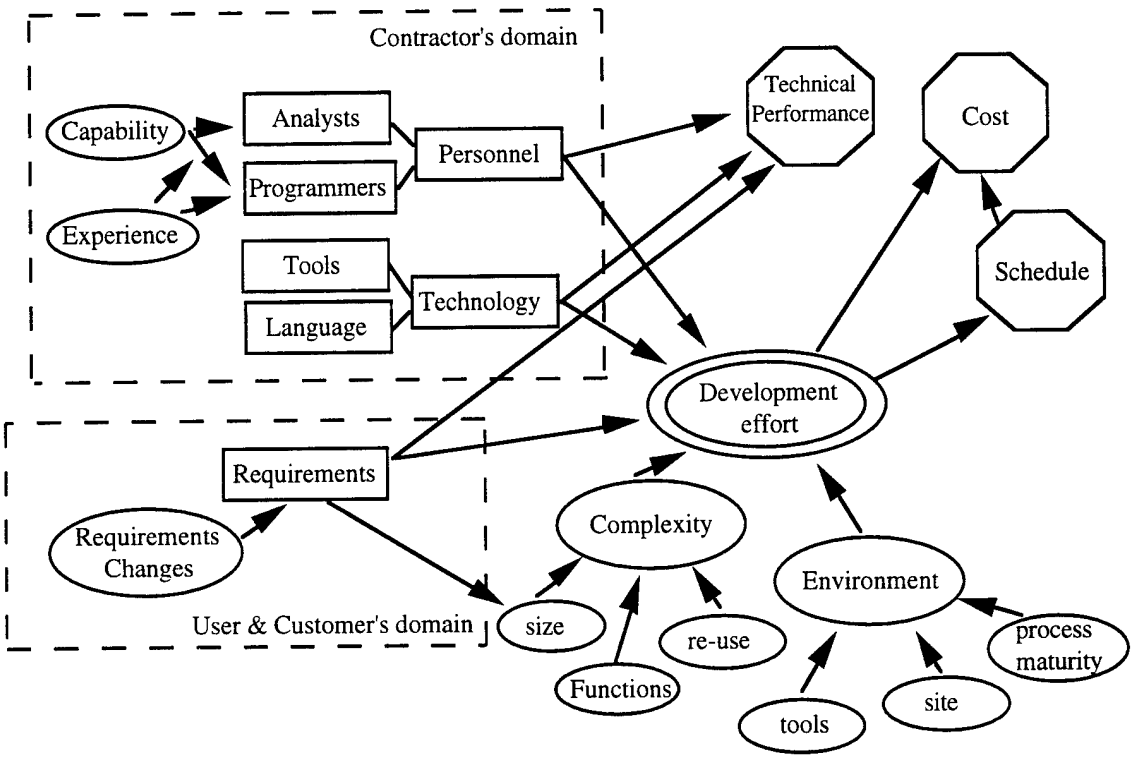


Figure 3.4 A representative influence diagram relationship of some HHM decomposition elements

3.5 Chapter Summary

This chapter has developed the HHM framework for software acquisition that provides a comprehensive framework for identification of risk sources and leads to the development of analytic models for the various software acquisition decompositions. The ultimate efficacy of the HHM framework lies in at least two dimensions: i) its capability to account for and display as complete a set (as possible) of sources of risk associated with software acquisition from their multidimensional perspectives as the analysts and experts can envision, and ii) its facility to provide in-depth varied interpretations of the various dimensions and relations of risks arising whether from the life cycle, program consequences, participants, project elements, or other perspectives.

The work of the following chapters builds on the HHM framework developed in this Chapter. In particular, the focus is on the ever-critical *program consequences* decomposition and the relationship between that vision and the *participant community* decomposition.

Chapter 4
Exact Determination of the Triangular Distribution's Conditional Expectations

Quantitative modeling of software acquisition decision-making situtations must account for the inherent uncertainties of the acquisition process and environment. Unfortunately, such decision situations often lack the objective data required for quantifying the uncertainty through the use of many common probability distributions. In such situations the triangular distribution is often employed [Haimes et al. 1994a]. Probabilistic analysis introduces the conditional expectation to be used in decision making as a measure of the risk of extreme events. Previous works focused on deriving the analytical expressions for the conditional expectation of extreme events for the normal, lognormal, and Weibull distributions [Leach and Haimes 1987], [Romei et al. 1992]. This Chapter extends previous results by deriving conditional expectation expressions for the triangular distribution and exploring the sensitivity of the conditional expectation of extreme events with respect to the selected probability partitioning point. The results are demonstrated in examples related to software acquisition project decision making.

4.1 Background

In most cases, the traditional use of a single value estimate in quantitative system modeling is not adequate to represent the variability associated with that system or environment. Because the assessment and ultimate prevention of risks (e.g., cost overrun) are of such major concern, it is desirable to retain as much information as is possible in the quantification of model parameters. Probabilistic quantification is one approach for addressing the problems associated with single value estimation. Unfortunately, in most cases where probabilistic quantification is practiced, the expected value is used as the sole measure of risk. As indicated in Section 2.6, this may lead to inaccurate or misrepresentative results.

The inadequacy of the expected value approach provides the motivation behind the development of the partitioned multiobjective risk method (PMRM) [Asbeck and Haimes 1984] with its conditional expected-value functions, or risk functions. In particular,

Asbeck and Haimes [1984] defined the following unconditional and conditional expectations of a random variable:

f_2 - high-probability, low-damage expectation
f_3 - intermediate damage and probability expectation.
f_4 - low-probability, high-damage expectation
f_5 - unconditional expectation.

In addition, there exists the additional objective function

f_1 - the cost function associated with implementing a mangement policy or risk mitigation effort.

The f_1 is often used in a multiple objective framework as the opposing objective to the minimization expectation functions.

Of the expectation functions listed above, the f_4 and f_5 values provide the broadest application to risk-based decision making. Extreme event analysis based on the f_4 and f_5 values has been applied to a wide variety of problems, including: water resources management [Karlsson and Haimes 1988], [Haimes and Karlsson 1989], [Haimes et al., 1992]; dam and flood management [Haimes 1986], [Haimes et al. 1998], [Li et al. 1992], [Lambert et al., 1994]; government project management [Haimes et al. 1994b]; and contractor selection [Haimes and Chittister 1993], [Haimes and Chittister 1995].

The f_2 and f_3 conditional expected values generally contribute less risk information for decision-making purposes than the f_4 and f_5 values. The f_2 may be useful in the rare cases where one is examining the most likely scenarios -- performance or behavior of the system in its most common lifecycle, excluding any extreme possibilities. The f_3, a restricted expected value calculation that also commensurates adverse events of high consequences and low probabilities with events of low consequences and high probabilities, is appropriate when the tail information of the distribution is unreliable, or for some other reason not pertinent to the analysis.

Our development and analysis of the expectations of the triangular distribution will focus on the f_4 and f_5 values. For completeness, the derivations of the f_2 and f_3 conditional expected values are included in Appendix C.

4.2 Previous Derivation of Conditional Expectation Equations

Leach and Haimes [1987] derived exact expressions for the conditional expectation of the normal distribution, while [Romei et al. 1992] contributed such expressions for the lognormal and Weibull distributions. Romei's work also explored the sensitivity of the conditional expectation with respect to distribution parameters, and the selected partitioning point.

The lack of data in many decision situations rules out the use of many probability distributions. In such situations, expert opinion is often sought to estimate a triangular distribution for the parameter. One of the simplest, yet still substantial probability distributions, the triangular distribution requires three parameters: the lower bound, the upper bound, and the most likely value. Use of the triangular distribution is widespread; the Army Corps of Engineers accepts the triangular distribution as the basis for their probabilistic analysis for flood and river risk management [Haimes et al. 1994a].

Extending the previous works to the triangular distribution will provide increased analytical support for decision makers in the situation of a lack of data.

4.3 Exact Determination of the Triangular Distribution Conditional Expectations

A triangular distribution has the following form of the probability density function [Law and Kelton 1982]:

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(b-a)(c-a)} & a \le x \le c \\ \dfrac{2(b-x)}{(b-a)(b-c)} & c \le x \le b \\ 0 & \textit{otherwise} \end{cases} \qquad (4.1)$$

where a is the minimum value, b is the maximum value, and c is the most likely value (Figure 4.1).

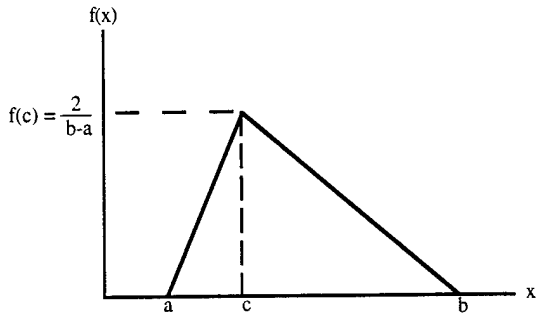


Figure 4.1 Triangular Probability Distribution

4.3.1 The Low-probability, High-damage Conditional Expectation, f_4

The high-consequence scenarios, those of greatest potential harm, are of most concern to decision makers. For that reason, the f_4 conditional expectation has particular significance in planning and policy evaluation. For a range of extreme events, the conditional expected damage (given that the damage is within that particular range) provides a measure of the impact associated with extremely large and potentially very costly and catastrophic events. The function f_4 is the expected value of the random variable, given that its observations are greater than a partitioning value β [Asbeck and Haimes 1984]:

$$f_4 = \mathrm{E}[X \mid x \geq \beta] = \frac{\int_\beta^\infty x f(x)dx}{\int_\beta^\infty f(x)dx}. \tag{4.2}$$

Deriving the conditional expectation for high-damage, low-probability events, f_4, for the triangular distribution is accomplished by substituting the element corresponding to $c \leq \beta \leq b$ of Eq. (4.1) into Eq. (4.2):

$$f_4 = \frac{\int_\beta^b \frac{2x(b-x)}{(b-a)(b-c)}dx}{\int_\beta^b \frac{2(b-x)}{(b-a)(b-c)}dx} = \frac{\int_\beta^b (bx - x^2)dx}{\int_\beta^b (b-x)dx}$$

$$= \frac{\left[\frac{1}{2}bx^2 - \frac{1}{3}x^3\right]_\beta^b}{\left[bx - \frac{1}{2}x^2\right]_\beta^b} = \frac{\left[\left(\frac{1}{2}b^3 - \frac{1}{3}b^3\right) - \left(\frac{1}{2}b\beta^2 - \frac{1}{3}\beta^3\right)\right]}{\left[\left(b^2 - \frac{1}{2}b^2\right) - \left(b\beta - \frac{1}{2}\beta^2\right)\right]}$$

$$= \frac{\frac{1}{6}b^3 - \frac{1}{2}b\beta^2 + \frac{1}{3}\beta^3}{\frac{1}{2}b^2 - b\beta + \frac{1}{2}\beta^2} = \frac{b^3 - 3b\beta^2 + 2\beta^3}{3(b^2 - 2b\beta + \beta^2)}$$

$$= \frac{(b-\beta)(b^2 + b\beta - 2\beta^2)}{3(b-\beta)^2} = \frac{(b-\beta)^2(b+2\beta)}{3(b-\beta)^2}$$

$$= \frac{b+2\beta}{3}.$$

Thus $$f_4 = \frac{b+2\beta}{3}, \quad c \le \beta \le b. \tag{4.3}$$

Hence, for the triangular distribution, the extreme event conditional expected value with damage partition β, $c \le \beta \le b$, is given by Eq. (4.3).

Applying Eq. (4.3) to the example problem described in [Haimes and Chittister 1993] verifies the expression (Table 4.1).

Table 4.1 Triangular Distribution Conditional Expectations Example data from [Haimes and Chittister 1993]

	Minimum (a)	Most Likely (c)	Maximum (b)	1-in-10 Extreme Event		1-in-100 Extreme Event	
				β	f_4*	β	f_4*
Customer	0.00	10.00	30.00	22.25	24.84	27.55	28.37
Contractor A	0.00	15.00	50.00	36.77	41.18	45.82	47.21
Contractor B	0.00	20.00	40.00	31.06	34.04	37.17	38.11

*calculated using equation (4.3), results compare exactly with [Haimes and Chittister 1993]

4.3.2 The Unconditional Expected Value, f_5

The general expression for the unconditional expectation, f_5, given by Asbeck and Haimes [1984] is:

$$f_5 = \frac{\int_{-\infty}^{\infty} x f(x)dx}{\int_{-\infty}^{\infty} f(x)dx}. \tag{4.4}$$

By definition of the probability density function, the denominator in Eq. (4.4) equals one, thus f_5 is equivalent to the common expected value $\mathrm{E}[x]$

$$f_5 = \int_{-\infty}^{\infty} xf(x)dx = \mathrm{E}[x]. \tag{4.5}$$

Deriving the unconditional expected value for the triangular distribution requires substituting Eq. (4.1) into Eq. (4.5):

$$f_5 = \left(\int_a^c xf(x)dx + \int_c^b xf(x)dx \right)$$

$$= \frac{1}{(b-a)(c-a)} \int_a^c (2x^2 - 2ax)dx \; + \frac{1}{(b-a)(b-c)} \int_c^b (2bx - 2x^2)dx$$

$$= \frac{1}{3(b-a)(c-a)} \left[2c^3 - 3ac^2 + a^3 \right] + \frac{1}{3(b-a)b-c)} \left[b^3 - 3bc^2 - 2c^3 \right]$$

$$= \frac{1}{3(b-a)c-a)} \left[(a-c)(a^2 + ac - 2c^2) \right] + \frac{1}{3(b-a)(b-c)} \left[(b-c)(b^2 + bc - 2c^2) \right]$$

$$= \frac{-(a^2 + ac - 2c^2)}{3(b-a)} + \frac{(b^2 + bc - 2c^2)}{3(b-a)}$$

$$= \frac{b^2 - a^2 + bc - ac}{3(b-a)} = \frac{(b-a)(b+a) + c(b-a)}{3(b-a)}$$

$$= \frac{(b-a)(b+a+c)}{3(b-a)}$$

$$= \frac{a+b+c}{3}.$$

This result is equivalent to the familiar expression for the expected value of the triangular distribution [Law and Kelton 1982]:

$$f_5 = \mathrm{E}[x] = \frac{a+b+c}{3}. \tag{4.6}$$

4.4 Sensitivity of the Triangular Distribution's $f_4(\cdot)$ Conditional Expectation with Respect to the Damage Partitioning Point

As f_4 has the most significance in risk analysis and in decision making, examination of the sensitivity of this conditional expected value with respect to the partitioning point is desirable. Of particular interest is the impact that changes in the probability partitioning value, α, have on the conditional expectation f_4. Recall that partitioning the probability axis of the exceedance function, 1-F(x), identifies the corresponding damage axis partitioning point β. For a given probability partition value, one would determine the damage partitioning point by solving

$$1 - F(\beta) = 1 - \alpha$$

or

$$F(\beta) = \alpha. \tag{4.7}$$

The exceedance function for the triangular distribution is adapted from the cdf [Law and Kelton 1982]:

$$1 - F(x) = \begin{cases} 1 & x < a \\ 1 - \dfrac{(x-a)^2}{(c-a)(b-a)} & a \le x \le c \\ \dfrac{(b-x)^2}{(b-a)(b-c)} & c \le x \le b \\ 0 & x > b \end{cases} \tag{4.8}$$

Considering β to be on the interval $[c,b]$, solving for Eq. (4.7) requires evaluating

$$\frac{(b-\beta)^2}{(b-a)(b-c)} = (1-\alpha)$$

which when solved for β results in the expression

$$\beta = b - [(1-\alpha)(b-a)(b-c)]^{1/2}, \quad \frac{c-a}{b-a} \le \alpha \le 1. \tag{4.9}$$

Eq. (4.9) is easily verified by applying the example from [Haimes and Chittister 1993] (Table 4.2).

Table 4.2 Triangular Distribution Partitioning Values Example
data from [Haimes and Chittister 1993]

	Minimum (a)	Most Likely (c)	Maximum (b)	Beta partitioning value*	
				alpha=.9	alpha=.99
Customer	0.00	10.00	30.00	22.25	27.55
Contractor A	0.00	15.00	50.00	36.77	45.82
Contractor B	0.00	20.00	40.00	31.06	37.17

*Calculated using equation (4.9); results exactly match [Haimes and Chittister 1993]

We can now state f_4 in terms of the probability partitioning value α by substituting the results from Eq. (4.9) into Eq. (4.3)

$$f_4 = \frac{b+2\beta}{3} = \frac{1}{3}\left(b+2\left(b-\left[(1-\alpha)(b-a)(b-c)\right]^{1/2}\right)\right)$$

or

$$f_4 = b - \tfrac{2}{3}\left[(1-\alpha)(b-a)(b-c)\right]^{1/2}, \quad \frac{c-a}{b-a} \le \alpha \le 1. \tag{4.10}$$

The conditional expected value f_4 depends on the selected probability partitioning value α and on the parameter vector $<a,b,c>$ of the distribution function. Facing uncertainty in the real world, especially where extreme events are concerned, the value of f_4 should be supplemented with its sensitivity analysis. Taking the derivative of Eq. (4.10) with respect to α

$$\frac{\partial f_4}{\partial \alpha} = -\tfrac{1}{3}\left[(1-\alpha)(b-a)(b-c)\right]^{-\frac{1}{2}}\left[-(b-a)(b-c)\right]$$

$$= \frac{(b-a)(b-c)}{3\left[(1-\alpha)(b-a)(b-c)\right]^{\frac{1}{2}}}$$

$$= \tfrac{1}{3}\left[\frac{(b-a)(b-c)}{(1-\alpha)}\right]^{1/2}. \tag{4.11}$$

The derivative (4.11) expresses the sensitivity of f_4 with regards to changes in the probability partitioning value. As $b \ge a$ and $b \ge c$, the numerator of Eq. (4.11) is nonnegative. Also, $0 \le \alpha < 1$, hence the denominator of Eq. (4.11) is on (0,1]. Therefore Eq. (4.11) is non-negative, implying f_4 is a non-decreasing function in α. The application of this result is shown in the following example problems.

4.5 Partitioning Sensitivity Examples

Two examples are presented to demonstrate probability partitioning sensitivity analysis for triangular distribution conditional expectations. The first example, adapted from [Haimes and Chittister 1993], is originally presented in the context of differentiating among the cost estimates from a customer and two contractor groups. Haimes and Chittister employed the conditional expected value in conjunction with the more common unconditional expected value, and we extend their results to indicate the significance of examining the probability partitioning value.

The second example also compares three cost estimates, each quantified in the form of a triangular probability distribution. In this example, each distribution has different high, low, and most likely parameter values, however all of the distributions have the same unconditional expected value. Sensitivity analysis concerning the selection of probability partitioning values for the conditional expectation is conducted to assist in decision making.

4.5.1 Example 4.1 -- Project Cost Overrun Evaluation

Consider the example proposed in [Haimes and Chittister 1993] where a governmental request for proposal (RFP) to develop a software system requires contractors to submit their estimate of the project's cost in a form that accounts for the variance to be expected in the project's cost overrun. Specifically, each contractor is asked to provide three values of the projected cost overrun percentage: (a) lower bound, (b) upper bound, and (c) most likely. The government (customer) also produces an estimate of the project's cost overrun. Table 4.3 indicates the parameters associated with each group's estimates, along with the unconditional expected value f_5 for each.

Table 4.3 Example 4.1 - Project cost overrun estimate parameters

	Minimum (a)	Most Likely (c)	Maximum (b)	Expected Value* (f5)
Customer	0.00	10.00	30.00	13.33
Contractor A	0.00	15.00	50.00	21.67
Contractor B	0.00	20.00	40.00	20.00

*Equation (4.6)

For comparison, Figure 4.2 depicts the triangular probability distributions for the customer and two contractors.

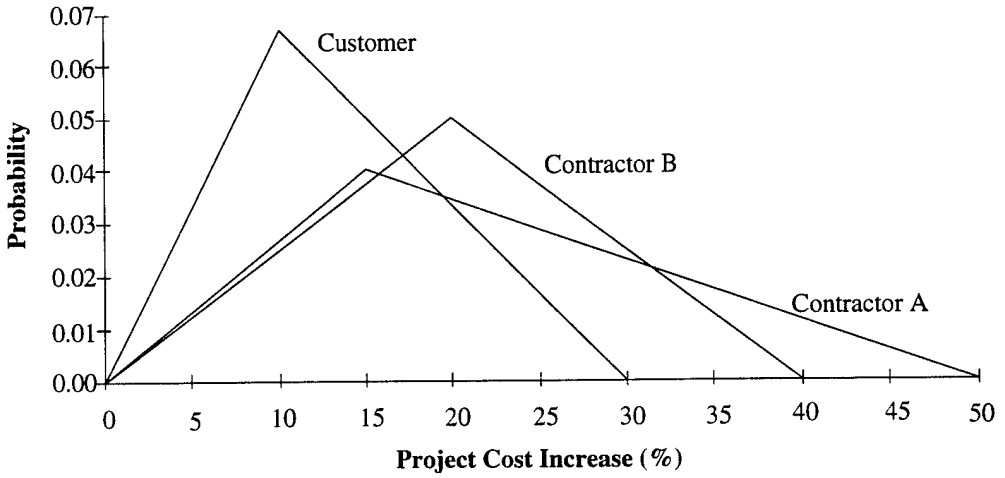


Figure 4.2 Triangular probability distributions for Customer and Contractors A and B

The low-probability, high-damage conditional expectation in terms of the probability partitioning value for each group's estimate is derived by substituting the values from Table 4.3 into Eq. (4.10). The f_4 equation for each is:

$$\text{Customer:} \quad f_4^{Cust} = 30 - \tfrac{20}{3}[6(1-\alpha)]^{1/2} \tag{4.12}$$

$$\text{Contractor A:} \quad f_4^{A} = 50 - \tfrac{10}{3}[70(1-\alpha)]^{1/2} \tag{4.13}$$

$$\text{Contractor B:} \quad f_4^{B} = 40 - \tfrac{20}{3}[8(1-\alpha)]^{1/2}. \tag{4.14}$$

A plot of Eqs. (4.12), (4.13), and (4.14) for various values α is shown in Figure 4.3. Observe the significant relative change in the conditional expected values between that of Contractor A and the other two groups as α increases. At $\alpha = 0.5$, the difference in the conditional expected value of percentage cost increase between Contractors A and B is approximately 3.5. However, at $\alpha = 0.9$, this difference has more than doubled to over 7.1. The various α partitioning values correspond to risk sensitivity levels for a decision maker. Higher α values permit greater focus on less-likely, yet more-damaging events.

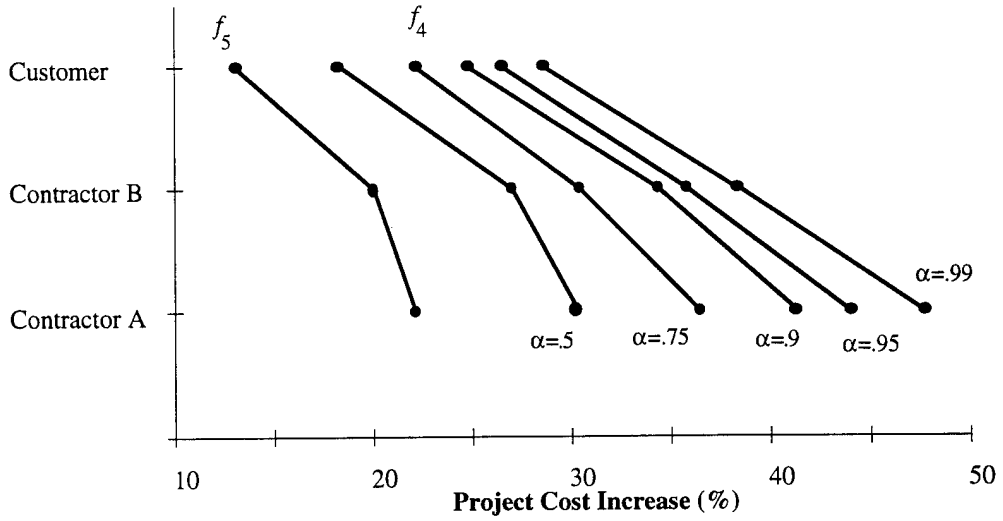


Figure 4.3 Unconditional and Conditional Percentage Cost Overrun for Varying α Values

Selection of the appropriate α value depends on the risk attitude of the decision maker. The selection can also be assisted by evaluating the sensitivity of the conditional expectation with respect to changes in α. Examining the rate of change of the conditional expected values as α increases (Eq. (4.11)), provides an indication of the risk involved with each estimate (Figure 4.4). In particular, one notices the rapidly increasing rate of change of the conditional expected value as the probability partitioning point is extended further into the extreme ranges. Seemingly miniscule changes in the partitioning value have dramatic effects on the conditional expectation. For example, extending the probability partitioning value from $\alpha = 10^{-4}$ to $\alpha = 10^{-5}$, increases the rate of change of the conditional expected value for the Customer from 816 to 2582! Extending the partitioning point further to $\alpha = 10^{-6}$ increases the conditional expected value's rate of change to 8165. The use of very small partitioning points is common for examining the extreme-scenario expected reliability of a mission-critical systems. For other problems such as the software cost problem, however, one is not necessarily concerned with exceptionally small partitioning values; examining the conditional expectation for $\alpha = 10^{-1}$ or $\alpha = 10^{-2}$ generally provides sufficient decision-making support. In these ranges, the conditional expected value is less sensitive to the partitioning point.

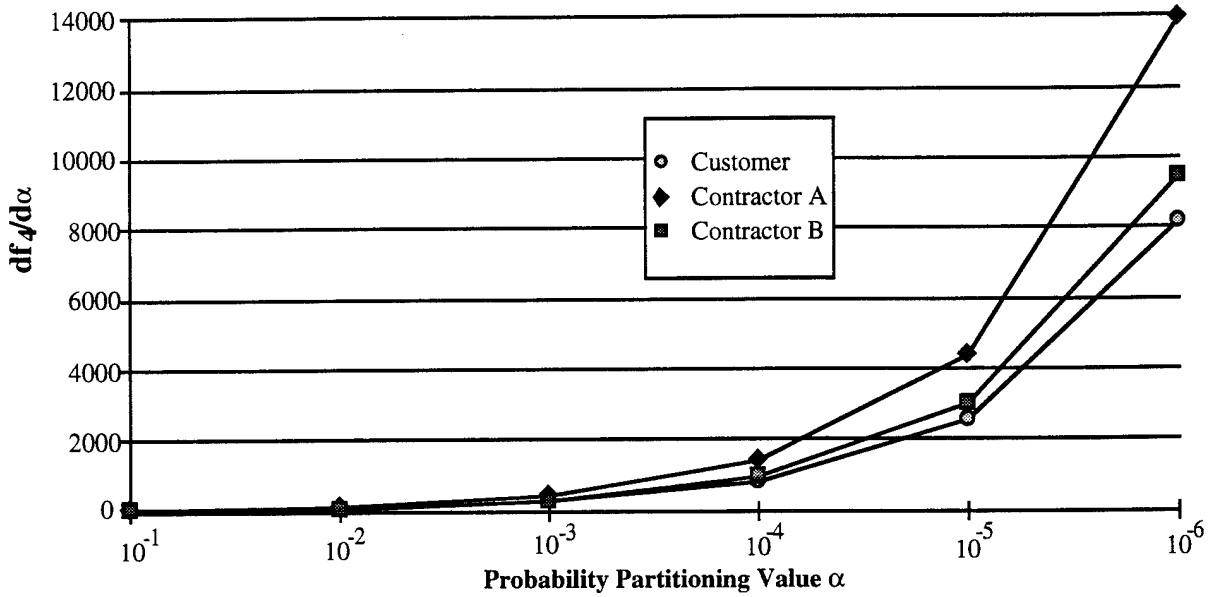


Figure 4.4 Rate of Change of f_4 for varying partitioning values

4.5.2 Example 4.2 -- Evaluating Alternatives with Identical Unconditional Expected Values

In this example, the triangular probability distributions of the cost of three alternatives (cases) are compared. While each case has the same unconditional expected value, the differences in conditional expected values at various probability partitioning points is significant. Table 4.4 provides a summary of the distribution parameters for each of the three cases.

Table 4.4 Example 4.2 - Cost estimate parameters for each case

	Minimum (a)	Most Likely (c)	Maximum (b)	Expected Value* (f5)
Case 1	10.00	30.00	50.00	30.00
Case 2	20.00	30.00	40.00	30.00
Case 3	5.00	10.00	75.00	30.00

*Equation (4.6)

Comparison of the three triangular probability distributions indicate wide differences in their variances (Figure 4.5).

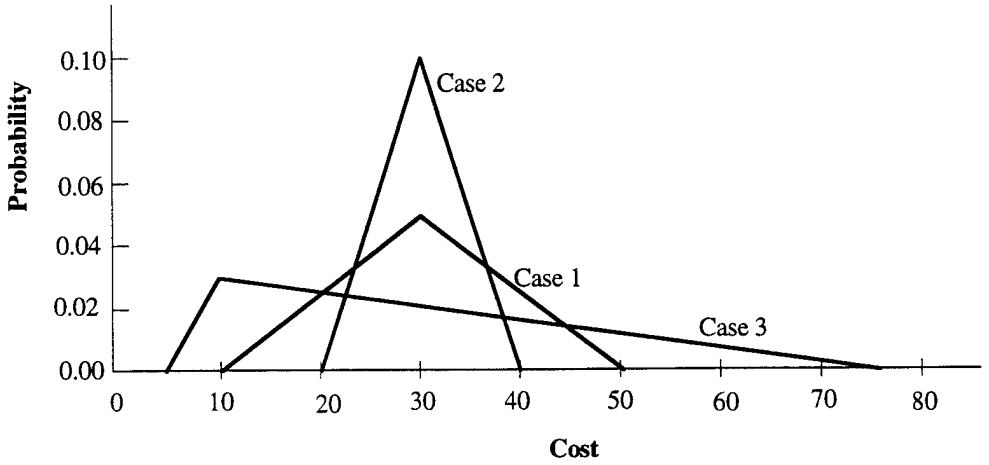


Figure 4.5 Triangular probability distributions for the three cases

Substituting the values from Table 4.4 into Eq. (4.10) produces the following f_4 equations for each case:

Case 1: $$f_4^{C1} = 50 - \tfrac{40}{3}[2(1-\alpha)]^{1/2} \tag{4.15}$$

Case 2: $$f_4^{C2} = 40 - \tfrac{20}{3}[2(1-\alpha)]^{1/2} \tag{4.16}$$

Case 3: $$f_4^{C3} = 75 - \tfrac{10}{3}[182(1-\alpha)]^{1/2}. \tag{4.17}$$

A plot of each case's values f_4^i (Eqs. (4.15), (4.16), and (4.17)) for various values of α is shown in Figure 4.6. Without the additional information of the conditional expectation, a decision maker may be indifferent among the three cases, as each has the same unconditional expected value. Again, the increasing difference in conditional expectation of the three cases as α increases provides a measure of each case's cost risk for extreme events. Examining the rate of change in the conditional expected value as α increases indicates the rapidly-increasing conditional expectation associated with Case 3, the most risky of the three cases (Figure 4.7).

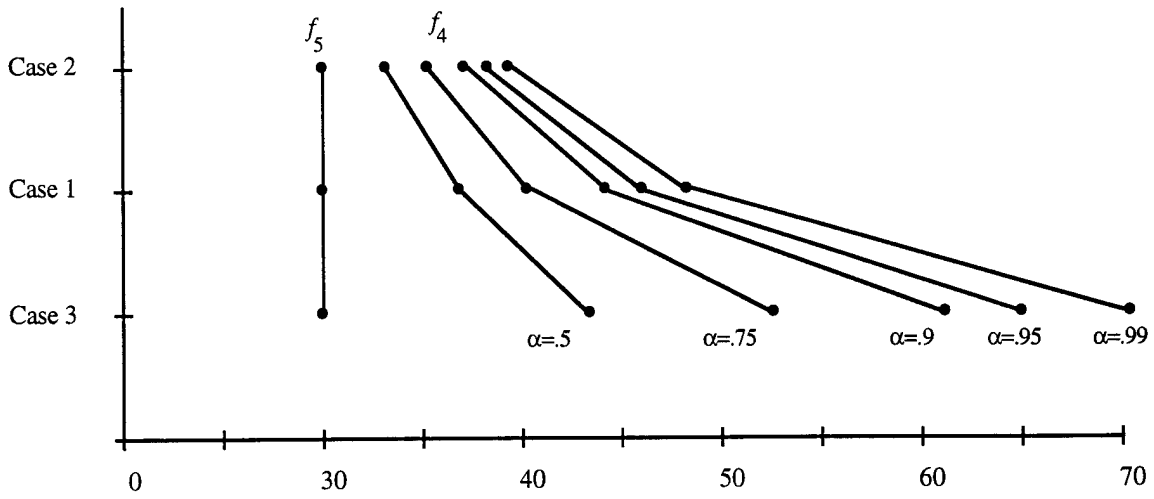


Figure 4.6 Unconditional and Conditional Expected Values for Varying α Values

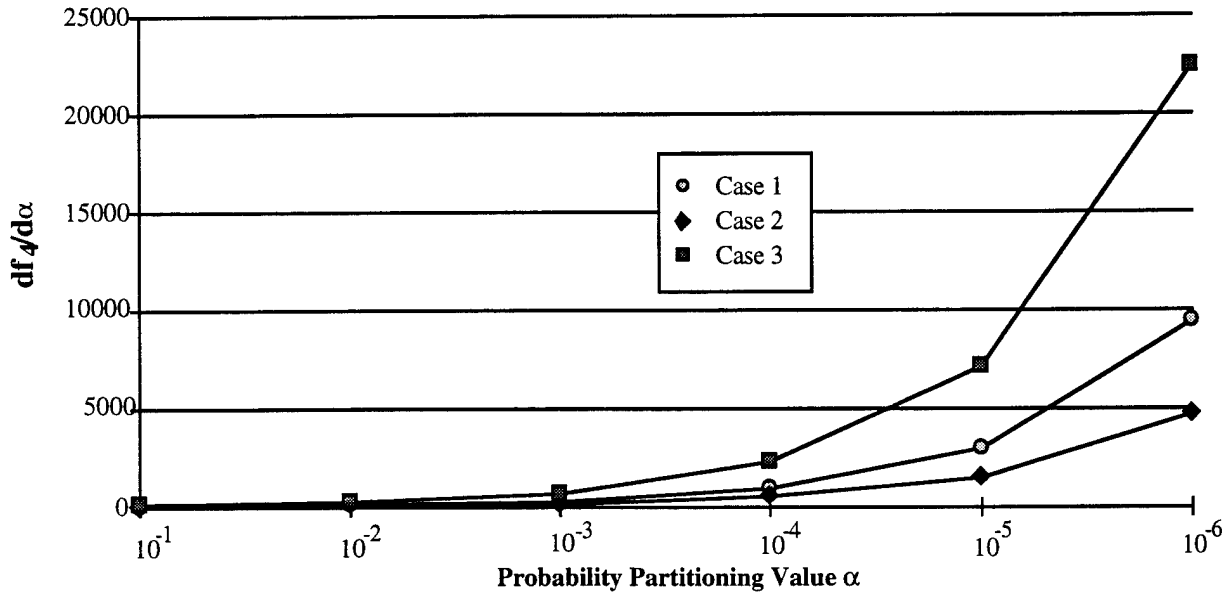


Figure 4.7 Rate of Change of f_4 for varying partitioning values

4.6 Chapter Summary

The additional information provided by the conditional expected value gives decision makers a better understanding of the risk of extreme events associated with alternative policy options. A supplement to the traditional expected value analysis, extreme event analysis provides greater insight regarding the impact of potential catestrophic events,

assisting decision makers in selecting appropriate risk-mitigation options. The contributions of this Chapter in deriving exact solutions of the triangular distribution's conditional expectation and partitioning sensitivity analysis with respect to partitioning values provides an increased analytical capability for evaluating decision situations that are hampered by a lack of empirical data. These results are applied within the methodologies developed over the next chapters.

Chapter 5
Probabilistic Software Estimation

This Chapter focuses on the *cost* element of the *program consequences* HHS of the HHM from Chapter 3. This Chapter develops a probabilistic approach to software estimation that focuses on the risk of extreme events and utilizes the conditional expected value as an additional risk management decision-making metric. The benefits that a probabilistic approach provide to project management are presented. A methodology for calculating the conditional and unconditional expected values from Monte Carlo simulation is developed. Application of the approach, using Monte Carlo simulation, is demonstrated for the Basic and Intermediate COCOMO models. Comparisons between the probabilistic approach, the original COCOMO results, and the actual project results are presented.

5.1 Introduction

One of the most difficult, yet important aspects of software project management is accurately estimating the needed resources for software development. Such estimation requires establishing the functions and performance characteristics of the desired system, estimating the size of the software product to be produced, estimating effort requirements, and producing project schedules.

Software estimation models are used to provide decision makers with a forecast of the actual manpower and time resources required to develop a software product. As these models are typically employed during the early phases of the development life cycle, accurately estimating a software project's resource requirements is complicated by the inherent uncertainty associated with quantifying the scope, size, and complexity of the project while still in these early stages. Most software estimation models rely on traditional, single-value parameter estimates, and produce a single-value estimate. Such an approach discounts the uncertainty associated with each parameter estimates and down-plays the uncertainty inherent in the early life cycle of project development .

While some probabilistic approaches to software estimation have been introduced (e.g., [Zhu and Lowther 1993]), their application has been limited due to proprietary restrictions and to a general lack of understanding within the software development community concerning how to effectively utilize such models.

5.2 Parameter Estimation Concerns for Software Estimation Models

Existing software estimation models rely on the use of historical data for gauging future program costs. Relationships among model variables (e.g., KLOC estimates, effort adjustment factors, development cost multipliers) are most often derived from historical experience using statistical regression techniques. These are then adjusted for a specific project's complexity or difficulty. Unfortunately, the rapid advancement in software practices, tools, and environments makes comparison to previous projects increasingly difficult [Przemieniecki 1993]. The validity and appropriateness of traditional estimation models are being called into question as to their applicability to today's software systems [Matson et al. 1994].

One of the limitations to increasing the accuracy of the KLOC-based models is the difficulty of estimating the number of lines of code that will be needed to develop a system from the information available at the requirements or design phases of development [Emrick 1987]. The KLOC-based models, to be useful, require one to be able to predict the size of the final product as early and accurately as possible. Unfortunately, estimating software size using the KLOC metric depends so much on previous experience with similar projects that different experts can make radically different estimates [Conte et al. 1986]. More detailed models require a greater number of variable and parameter estimates, each of which contributes to the overall uncertainty of the final effort and schedule estimates. The Detailed COCOMO Model requires one to estimate 15 cost multipliers for each of the 4 development phases -- a total of 60 parameter estimates!

The function-point approach is no less immune from a large number of required estimates. A function point count requires 15 estimates -- an estimate of the count of each component (input, output, files, interfaces, and inquiries) at each level (low, average, high) [Albrecht and Gaffney 1983]. Each estimate is subject to a range of possible values. Within a single organization there are sometimes significant differences in the function point counts for the same project as determined by separate individuals [Matson et al. 1994]. This arises from subjective assessments in both the raw counts and adjustment factors.

5.3 Accounting for Uncertainty in Software Estimation

Recognizing that the result of a software estimation model relies on the subjective assessment of numerous model parameters, a means to account for, and describe, the uncertainty of the estimate

is required. The common practice of many organizations is simply to conduct multiple estimations -- with separate groups of personnel preparing independent estimates [Barrow et al. 1993]. The drawback of this approach is the task of resolving the differences in estimates. Unfortunately, this approach often causes that the organization accepts the average of all proposals or some other more politically-motivated means as its strategy for determining the final estimate.

Without addressing an estimation model in particular, Pressman [1987] introduced an approach to account for software cost uncertainty, suggesting the use of three estimates for each parameter. The most likely (m) is, as its name suggests, the best guess assuming nothing goes wrong. The other two estimates are an optimistic (o) and a pessimistic (p) one. Pressman then advocates using the expected (e) estimate, computed as

$$e = (o + 4m + p)/6.$$

The formula adjusts the estimate by approximating the fact that, in the normal distribution, one third of the points will be more than one standard deviation from the mean.

Pressman's approach may allow for an initial consideration of an estimate's variability; however, reducing the three values to a single expected estimate discards the additional information contained in the range of the estimates. An improved approach would retain and utilize the information gained through estimating o, m, and p.

Haimes and Chittister [1993], [1995] address the issue of maintaining information, also proposing the use of three estimates: most likely, high, and low. The application of their approach would require a software developer to submit three cost estimates in place of the more common single value when bidding on a contract. Unlike Pressman's approach, however, the three estimates are employed as the basis for forming a triangular probability distribution for the project's cost. The use of a probability distribution not only maintains the information concerning the most likely scenario, but also provides an indication of the uncertainty associated with that estimate. Additional decision-making information and risk information are available with this approach. The principal shortcoming of this approach is that it estimates cost directly, without explicitly capturing the many factors included in traditional software estimation methodologies.

Zhu and Lowther [1993] describe a probabilistic software estimation approach for the Intermediate COCOMO Model. Their approach, implemented in a spreadsheet model, allows the cost drivers, KLOC estimate, cost-per-effort, and other parameters of the COCOMO model to be represented by probability distributions. A Monte Carlo simulation of the realizations of the model's input

parameter probability distributions is used to produce a distribution of the development effort. Unfortunately, their paper does not address the benefits of such an approach, how one would make use of the probabilistic aspects of the model, or how the information provided in the model is to be used for decision making.

The remainder of this Chapter is devoted to the development of a probabilistic approach to software estimation that builds on the works of Haimes and Chittister, and Zhu and Lowther. This approach is based on the risk of extreme events and utilizes the conditional expected value [Asbeck and Haimes 1984] as an additional decision-making metric.

5.4 The Probabilistic Software Estimation Approach

The central concept of the probabilistic approach to software estimation is the explicit incorporation of uncertainty. Where possible, all or any of the model's parameters are quantified in terms of probability distributions. Those parameters not so quantified are set to their expected value. Applying a software estimation model, the output is now in terms of a probability distribution instead of the traditional single-point value. To demonstrate the approach, we consider two application contexts. First, the direct approach for circumstances in which the development effort is directly quantified by a known, closed-form probability distribution. For this case, the conditional and unconditional expected values can be determined by Eqs. (4.2) and (4.4) (or Eqs. (4.3) and (4.6) for the triangular distribution). Second, a Monte Carlo simulation approach for situations where the development effort cannot be quantified by a closed-form distribution. In such cases, we can estimate the expectation values by examining the set of outcomes of the simulation. Examples of each approach are presented, demonstrating the use of the conditional expected value for decision making.

5.4.1 Direct Approach to Probabilistic Software Estimation

Quite often, an initial evaluation of the effort required to develop a software product can be estimated, not by way of a traditional estimation model, but by the direct assessment of a probability distribution of the development effort . The expected value and conditional expected values of this distribution are then used for evaluating the relative desirability of various program alternatives. Romei [Romei et al. 1992] derived the exact-form solutions for the conditional and unconditional expectation functions for normal, lognormal, and Weibull distributions. Unfortunately, there is often insufficient empirical evidence to support the use of one of these

distributions for estimating software development effort. In such cases, subjective assessment of probability distributions by area experts using the triangular distribution or fractile method [Haimes and Chittister 1993] is appropriate. The triangular distribution expectation function results of Chapter 4 are central to the direct approach to software estimation. The direct approach is demonstrated in the following example.

5.4.1.2 Example 5.1 - Alternative Selection using the Direct Approach. Consider the project manager who is debating about the selection from among three alternative approaches for a software system. Expert evaluation of each alternative led to the triangular distribution information in Table 5.1 concerning the most likely, lowest, and highest development effort requirements. While each alternative has the same unconditional "business as usual" expected value, E[X], the differences in conditional expected values at various probability partitioning points are significant. This comparison also indicates wide differences in their variances (Figure 5.1).

Table 5.1 Development effort (man-month) estimates for each alternative

	Minimum (a)	Most Likely (c)	Maximum (b)	Expected Value (f5)
Alternative 1	10.00	30.00	50.00	30.00
Alternative 2	20.00	30.00	40.00	30.00
Alternative 3	5.00	10.00	75.00	30.00

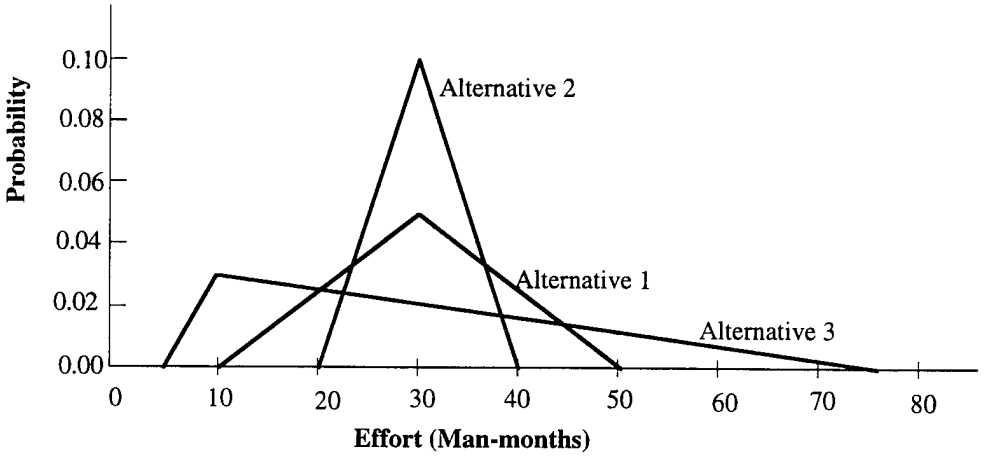


Figure 5.1 Development effort probability distributions for the three alternatives

The conditional expected value, f_4, of the triangular distribution can be stated in terms of the probability partitioning values β and α Eqs. (4.3) and (4.10):

$$f_4 = \frac{b+2\beta}{3}, \quad c \le \beta \le b \tag{5.1}$$

and

$$f_4 = b - \tfrac{2}{3}\left[(1-\alpha)(b-a)(b-c)\right]^{1/2}, \quad \frac{c-a}{b-a} \le \alpha \le 1. \tag{5.2}$$

The value f_4 depends on the selected probability partitioning value α and on the parameter vector $<a,b,c>$ of the triangular distribution function. Substituting the values from Table 5.1 into Eq. (5.2) produces the following conditional expected-value functions for the three alternatives:

Alternative 1: $f_4^{A1} = 50 - \tfrac{40}{3}\left[2(1-\alpha)\right]^{1/2}$

Alternative 2: $f_4^{A2} = 40 - \tfrac{20}{3}\left[2(1-\alpha)\right]^{1/2}$

Alternative 3: $f_4^{A3} = 75 - \tfrac{10}{3}\left[182(1-\alpha)\right]^{1/2}$.

A comparative plot of f_5 and f_4 for each alternative for various α values is shown in Figure 5.2.

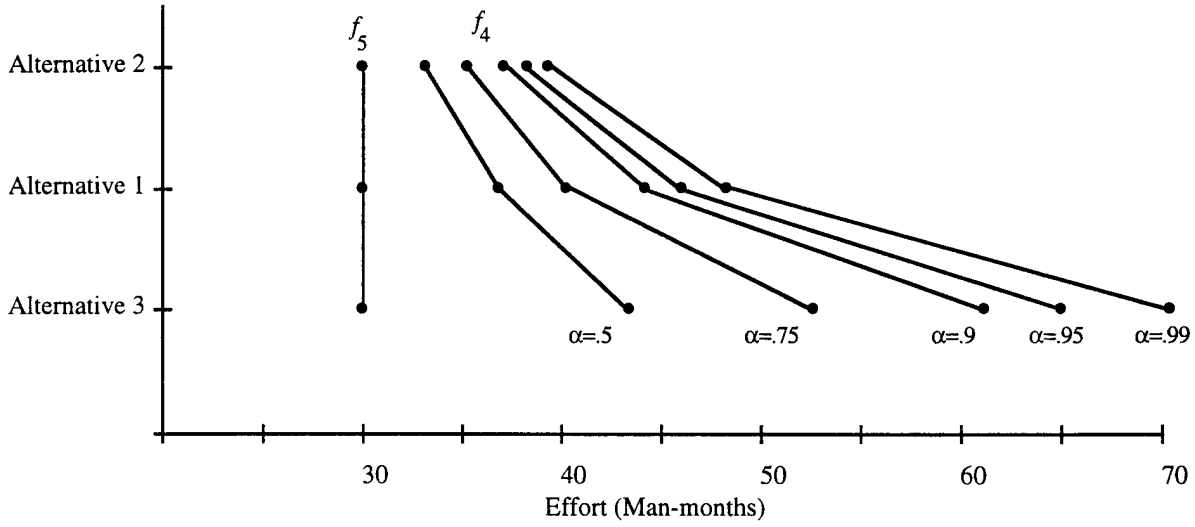


Figure 5.2 Unconditional and conditional expected values for varying α values

Without considering the additional information of the conditional expected value, each alternative seems equally desirable -- each has the same unconditional expected value. However, given the additional information of the conditional expectation, we observe the increasing difference in the three alternatives as α increases. Compare, for example, the conditional expected values of the three alternatives for $\alpha = 0.9$, the 1-in-10 scenario. Alternative 2's conditional expected value is 37.02 man-months, a 23 percent increase over the unconditional expected value. The 1-in-10

conditional expected value of Alternative 1 is 44.04 man-months, while that of Alternative 3 is 60.78 -- more than double its unconditional expected value!. We see that Alternative 2 is the least risky of the three alternatives, while the greatly-increasing conditional expected value of Alternative 3 indicates the influence of the tail of its distribution. The conditional expectation provides a measure of each alternative's development effort risk for extreme events.

5.4.2 Monte Carlo Simulation Approach to Probabilistic Software Estimation

While the previous method developed a direct probabilistic estimate of project development effort, this approach considers the use of a traditional, analytic cost estimation model, for which all or some of the input parameters are quantified by probability distributions. For example, consider the probabilistic estimation of project development effort using the Intermediate COCOMO model Eq. (2.3). If we treat the input parameter KLOC as a random variable (a triangular distribution, for instance), then evaluating Eq. (2.3) requires solving a nonlinear function of a random variable. The result will be a probability distribution of the development effort, but one without explicit, analytic expression.

While there are analytic, closed-form solutions for the function of certain random variables having known distributions (e.g., a linear combination of normal random variables produces a normal random variable), such is not the case for a nonlinear function of a triangular random variable. In this case we must apply an approximation method -- Monte Carlo simulation -- to determine the distribution of Eq. (2.3) and evaluate its conditional and unconditional expected values.

With Monte Carlo simulation, the outcome of probabilistic events is determined by randomly drawing one value from each parameter's density function and assessing the outcome based on those random draws. A single outcome is only one sample out of a very large number of possible cases. In order to discover what the expected or average outcome of the situation would be, it is necessary to run many cases -- take a large sample -- varying only the random values selected.

To simplify notation, denote man-months of development effort by M (instead of the previous MM). To approximate $E[M]$, the expected man-months of development effort, we generate a random value $KLOC^{(1)}$= kloc from the density function of KLOC and then compute $M^{(1)}$ by Eq. (2.3). We next generate a second random value (independent of the first) $KLOC^{(2)}$ and compute $M^{(2)}$. This continues until n, a fixed number of independent and identically distributed random variables $M^{(i)} = (EAF)(a(KLOC^{(i)})^b)$, $i = 1, \ldots, n$ have been generated. By the strong law of large numbers [Ross 1989] we know that

$$\lim_{n\to\infty}\frac{M^{(1)}+\cdots+M^{(n)}}{n}=E[M]=f_5. \tag{5.3}$$

Hence, we can use the average of the generated $M^{(i)}$s as an estimate for $E[M]$, the unconditional expected value. To approximate the conditional expected value, f_4, we form the sub-set of outcomes M_β whose members are those outcomes that exceed the partitioning value β associated with the predetermined α value. Simply stated,

$$M_\beta=\left\{M^{(i)}\middle|M^{(i)}\geq\beta\right\}. \tag{5.4}$$

The average of the members of M_β is the average of all outcomes that exceed a particular damage level -- precisely the definition of the conditional expected value. Therefore, given m elements of the set M_β (for m sufficiently large), an approximation to the conditional expected value of the development effort is given by

$$\frac{M_\beta^{(1)}+\cdots+M_\beta^{(m)}}{m}\cong E[M_\beta]=E[M|M\geq\beta]=f_4. \tag{5.5}$$

Obviously, Monte Carlo simulation is a computationally-intensive approach, as the number of observations n must be large. Fortunately, the availability of personal computers and powerful, yet easy-to-use simulation software packages makes this approach feasible. Without such resources, the method would be computationally burdensome. In the following example, we use the software package @RISK [Palisade 1995] to perform the Monte Carlo simulation calculations for the Intermediate COCOMO model.

5.4.2.1 Example 5.2 - The Monte Carlo Approach for Intermediate COCOMO.

For this example, we extend the problem described in [Boehm 1981] and evaluate the required development effort of three competing alternatives for a semidetached software project. Unlike the first example, quantification of the required development effort is achieved using the Intermediate COCOMO model Eq. (2.3). The cost multipliers that constitute the effort adjustment factor (EAF) are assumed to be identical for each alternative, where EAF = 1.18. For a semidetached product, the parameters of Eq. (2.3) are set at a = 3.0, b = 1.12 (see Table A.3). The KLOC estimate for each alternative is quantified as a triangular probability distribution (Table 5.2). With an expected value of 32 KLOC, the deterministic expression of Alternative 2 is identical to the original problem in [Boehm 1981].

Table 5.2 KLOC requirement estimates for each alternative

	Minimum (a)	Most Likely (c)	Maximum (b)	Expected Value* (f5)
Alternative 1	20	30	48	32.67
Alternative 2	26	32	38	32.00
Alternative 3	28	32	46	35.33

A Monte Carlo simulation of 1000 iterations (i.e., $n = 1000$) using the approach described above was conducted for each of the three alternatives of Table 5.2. Figure 5.3, a histogram plot of the simulation results for Alternative 2, provides a graphical representation of the simulation's approximation of the output probability distribution for development effort. The partitioning point for determining the conditional expected value was set at $\alpha = 0.9$, the 1-in-10 occurrence. The conditional and unconditional expected value results for each alternative are given in Table 5.3. Since the parameters associated with Alternative 2 were selected to be identical to the example from [Boehm 1981], we can compare our results with the original paper. The expected development effort result from the simulation for Alternative 2, $f_5 = 172.05$, is nearly identical to Boehm's reported 172.

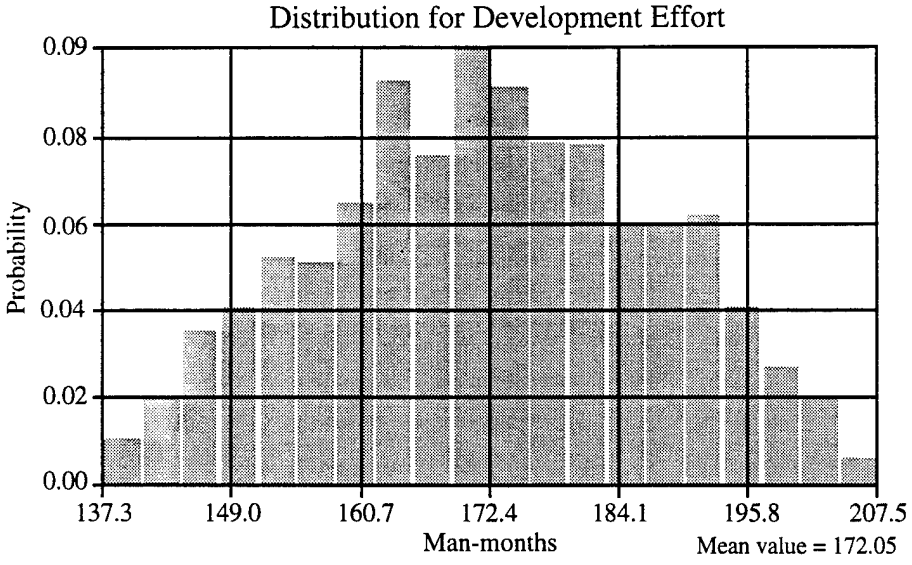


Figure 5.3 Histogram of Monte Carlo Simulation Results for Alternative 2

Table 5.3 Expected Value Results from Monte Carlo Simulation

	Expected Value	
	f_5	f_4 ($\alpha = 0.9$)
Alternative 1	176.19	242.55
Alternative 2	172.05	198.31
Alternative 3	192.30	238.09

A plot of the f_4 and f_5 results provides a graphical comparison of the desirability of each alternative (Figure 5.4). While the unconditional expected values of Alternatives 1 and 2 are very close, there is great disparity in their 1-in-10 extreme-event, conditional expected values. Without the additional information of the conditional expected value, a decision maker may be indifferent between Alternatives 1 and 2. However, the conditional expected value indicates that Alternative 2 is the least risky of the three alternatives. If all three alternatives were to have the same implementation costs, Alternative 2 would be most preferable. For a complete evaluation of the relative preference of one alternative toward another, one must determine the cost of each alternative, and then tradeoff this cost with the conditional and unconditional expected development effort values of each alternative.

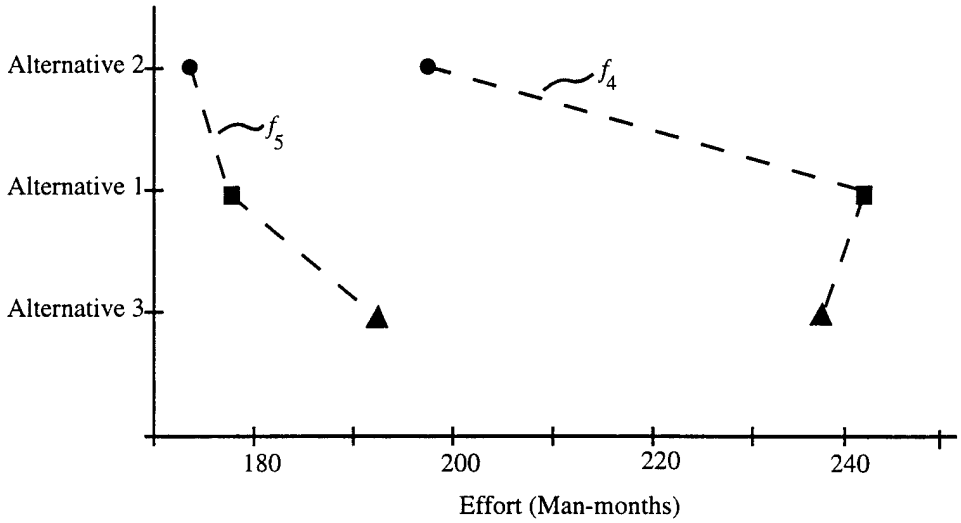


Figure 5.4 Unconditional and conditional expected values from Monte Carlo simulation

5.5 Comparing Probabilistic Results, Original Model Results, and Actual Values

In the previous sections we demonstrated the added insight that is provided to decision makers through the use of a probabilistic approach to software cost estimation. The additional information of the conditional expected value contributes an improved understanding of the risk associated with

each alternative. We are now interested in comparing the results of the probabilistic approach with the results of the original model. Note that both comparisons use the actual project development effort from the software project data introduced in Boehm's original data set [Boehm 1981] (Table 5.4). The original data for each software development project include: classification of the development mode (organic, semidetached, embedded), the total KLOC of the delivered product, the effort adjustment factor (EAF), the project's actual development effort in man-months, and the estimated effort results of the original Basic COCOMO model and the Intermediate COCOMO model.

It is important to note that the KLOC parameter in the data set represents the actual number of lines of code that were developed for the software project. This is a critical observation, for in practice, the use of such ex-post data is never possible. Software estimation models are employed early in the development life cycle, requiring one to use an ex-ante estimate of the required KLOC -- not the actual total -- to estimate project cost. Unfortunately, Boehm's data set of software projects does not contain ex-ante KLOC requirement estimates.

Estimates of KLOC requirements made early in the development life-cycle often differ greatly from the actual final total. As Augustine states [Augustine 1993], "in 90% of the cases, cost is underestimated from the beginning." This statement applies to KLOC estimates, which can be considered a pseudo-cost variable. While original ex-ante estimates of the KLOC requirements for the projects of Table 5.4 do not exist, we use the actual ex-post data as the basis for a probabilistic formulation of KLOC requirements, keeping in mind Augustine's observation.

5.5.1 Baseline Comparison Formulation

In order to compare the probabilistic approach with the original deterministic method, we apply the Monte Carlo simulation approach to the Basic and Intermediate COCOMO models to determine the values of unconditional and conditional expected development effort for each project in the dataset. Although the KLOC values in Table 5.4 are the exact, ex-post values, we generously treat them as extremely accurate ex-ante estimates made by software development experts. In recognition of the inherent uncertainty in estimating a project's size, the KLOC input for each project is quantified in terms of a triangular probability distribution.

For each software project, the three triangular distribution parameters are set at: a = 0.5*(actual KLOC), b = 1.75*(actual KLOC), and c = 0.75*(actual KLOC). This approach slightly understates the actual KLOC requirement, yet maintains the actual value as the expected value for

Table 5.4 Software Development Projects Data, from [Boehm 1981]

					Estimated Effort (MM)	
Project	Mode*	Actual KLOC	EAF	Actual Effort(MM)	Basic COCOMO	Intermed. COCOMO
1	3	113.00	2.72	2040.00	1047.00	2218.00
2	3	249.00	0.84	1600.00	2702.00	1770.00
3	2	132.00	0.34	243.00	711.00	245.00
4	1	46.00	1.17	240.00	134.00	212.00
5	1	16.00	0.66	33.00	44.00	39.00
6	1	4.00	2.22	43.00	10.30	30.00
7	1	6.90	0.40	8.00	18.00	9.80
8	3	22.00	7.62	1075.00	147.00	869.00
9	3	30.00	2.39	423.00	213.00	397.00
10	3	18.00	2.38	321.00	115.00	214.00
11	3	20.00	2.38	218.00	131.00	243.00
12	3	37.00	1.12	201.00	274.00	238.00
13	3	24.00	0.85	79.00	163.00	108.00
14	2	3.00	5.86	73.00	10.30	60.00
15	3	3.90	3.63	61.00	18.00	52.00
16	3	3.70	2.81	40.00	17.00	38.00
17	3	1.90	1.78	9.00	7.80	10.70
18	3	320.00	3.89	11400.00	3652.00	11056.00
19	3	966.00	0.73	6600.00	13749.00	7764.00
20	2	287.00	3.85	6400.00	1698.00	6536.00
21	3	252.00	0.86	2455.00	2741.00	1836.00
22	3	109.00	0.94	724.00	1003.00	733.00
23	3	75.00	0.89	539.00	640.00	443.00
24	2	90.00	0.70	453.00	463.00	326.00
25	3	38.00	1.95	523.00	283.00	430.00
26	3	48.00	1.16	387.00	375.00	339.00
27	3	9.40	2.04	88.00	53.00	89.00
28	1	13.00	2.81	98.00	35.00	133.00
29	2	2.14	1.00	7.30	7.00	7.00
30	2	1.98	0.91	5.90	6.40	5.80

*Mode: 1 = Organic 2 = Semidetached 3 = Embedded

each distribution. All other model parameters are fixed at their original values. Deterministic evaluation of the Basic and Intermediate COCOMO models, using the expected values of KLOC in the effort equations, produces the same results as those of the original models (with slight variation due to rounding in [Boehm 1981]). Since each input distribution was developed with an expected

value identical to the actual KLOC value, we anticipate the resulting unconditional expected value (f_5) of the probabilistic approach to be relatively close to the forecast of the original model.

5.5.1.1 Baseline Comparison: Basic COCOMO. As the goal of the software estimation models is to accurately predict the actual development effort and costs, we compare the model forecasts to the actual development requirements. A comparison of Boehm's original Basic COCOMO model with that of the actual project development effort, along with similar plots of the probabilistic results, f_5 and f_4, is shown in Figure 5.5.

Due to the conservative approach in using the actual KLOC requirement as the expected value of the input distribution, Figures 5.5(a) and 5.5(b) are nearly identical -- a result that was expected. Such an observation validates the probabilistic approach as being consistent with the original COCOMO model. Interestingly enough, the conditional expected value also provides a consistent measure of actual development effort (Figure 5.5(c)). This, in part, is an effect of the limited variability of the input distributions. The conditional expected value may often be a more accurate indicator of actual development effort, particularly considering the tendency of underestimating parameters early in the life cycle. From these comparative results, one realizes that the probabilistic approach not only provides additional information concerning the extreme event scenarios of an alternative, but its results are consistent with those of the original method.

To better evaluate the model results as they compare to the actual values, we calculate the normalized percentage error for each project in the dataset. Plots of the normalized percentage error histograms depict the range of accuracy with which the original model and its probabilistic extensions forecast actual development effort requirements. A distribution with the majority of its results centered around the zero value indicates an accurate predictive model. From Figures 5.6(a) and 5.6(b), one observes the tendency of the Basic COCOMO model to underestimate development effort, even when the actual KLOC value is used in the model. The distribution of normalized error associated with the conditional expected value is shown in Figure 5.6(c). As it is an indicator of the extreme event scenarios, one would anticipate that the conditional expected value would tend to overestimate the actual value. Figure 5.6(c) exhibits some evidence of overestimation, however not as great as might have been expected.

5.5.1.2 Baseline Comparison: Intermediate COCOMO. Conducting Monte Carlo simulation of the Intermediate COCOMO model for the projects in the dataset produces results consistent with that of the original model (Figure 5.7). Comparing Figure 5.7 with Figure 5.5

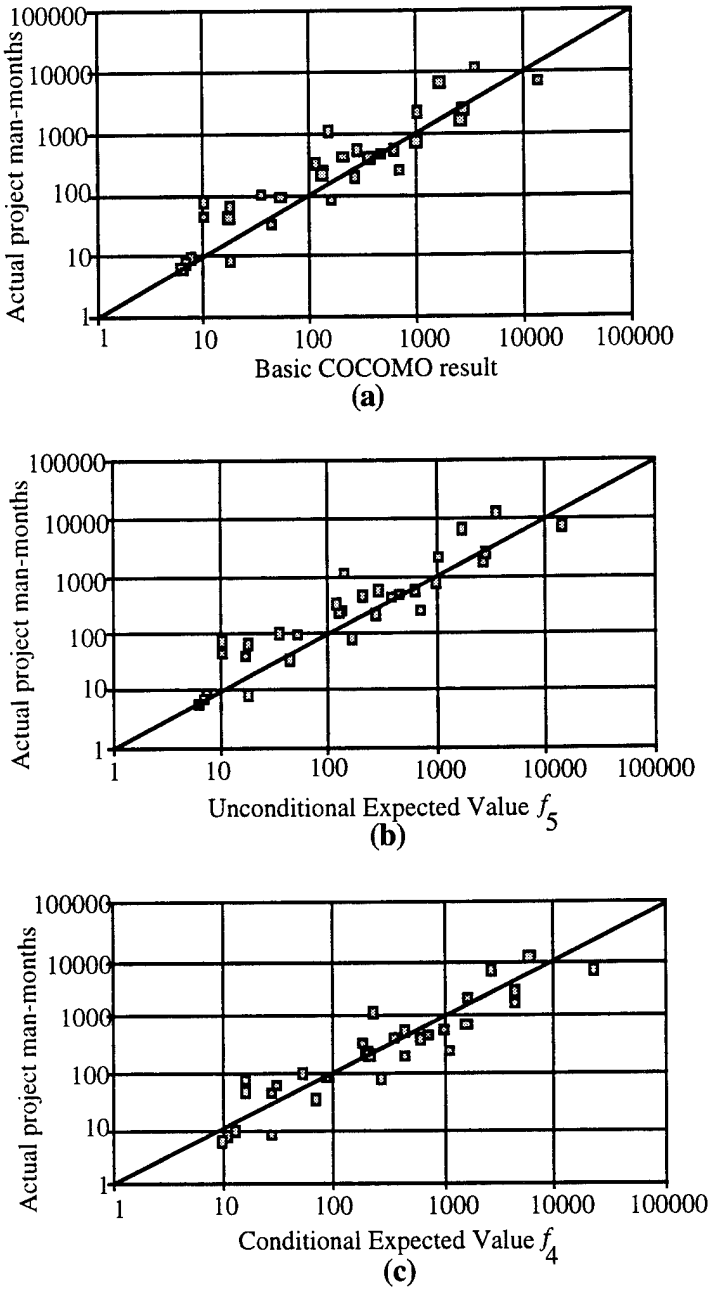


Figure 5.5 Basic COCOMO model results: (a) Original forecast versus actuals, (b) Unconditional expected values versus actuals, (c) Conditional expected values versus actuals.

shows the Intermediate COCOMO model's improvement over the Basic model in accurately forecasting actual development effort. Again, because of the conservative scheme employed for developing the input distributions, the unconditional expected value results (Figure 5.7(b)) are similar to the original results (Figure 5.7(a)). Examining Figure 5.7(c), we observe that the conditional expected values tend to over-estimate the actual development effort. While this is true,

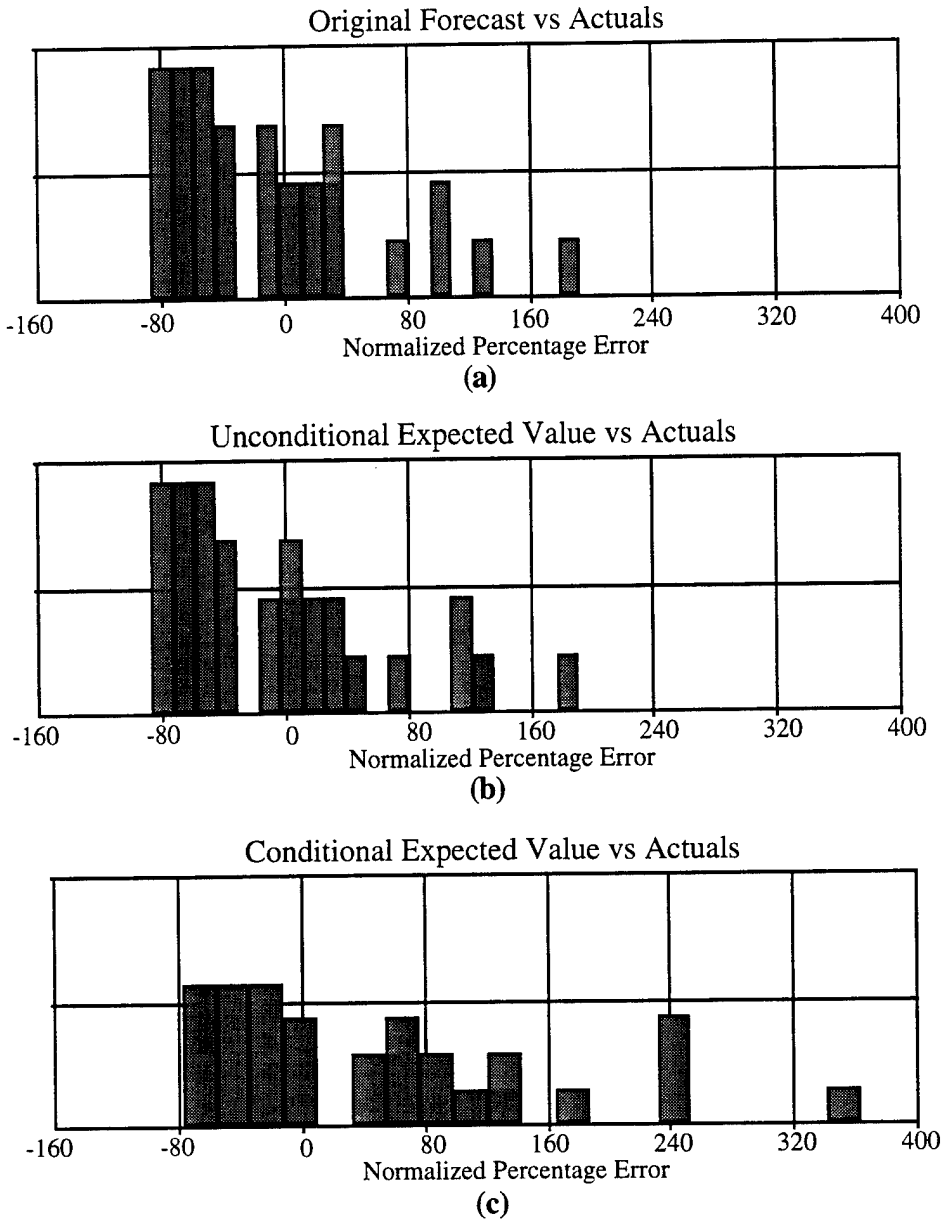


Figure 5.6 Baseline Comparison: Basic COCOMO model normalized percentage error: (a) Original forecast versus actuals, (b) Unconditional expected values actuals, (c) Conditional expected values versus actuals.

it is readily apparent that the f_4 values still show good correlation with the actual development effort values.

In a more-realistic application, where the uncertainty concerning the project's size early in the software development life cycle causes underestimation of the KLOC requirements, the conditional expected value may be an important complementary metric. Conducting analysis similar to the

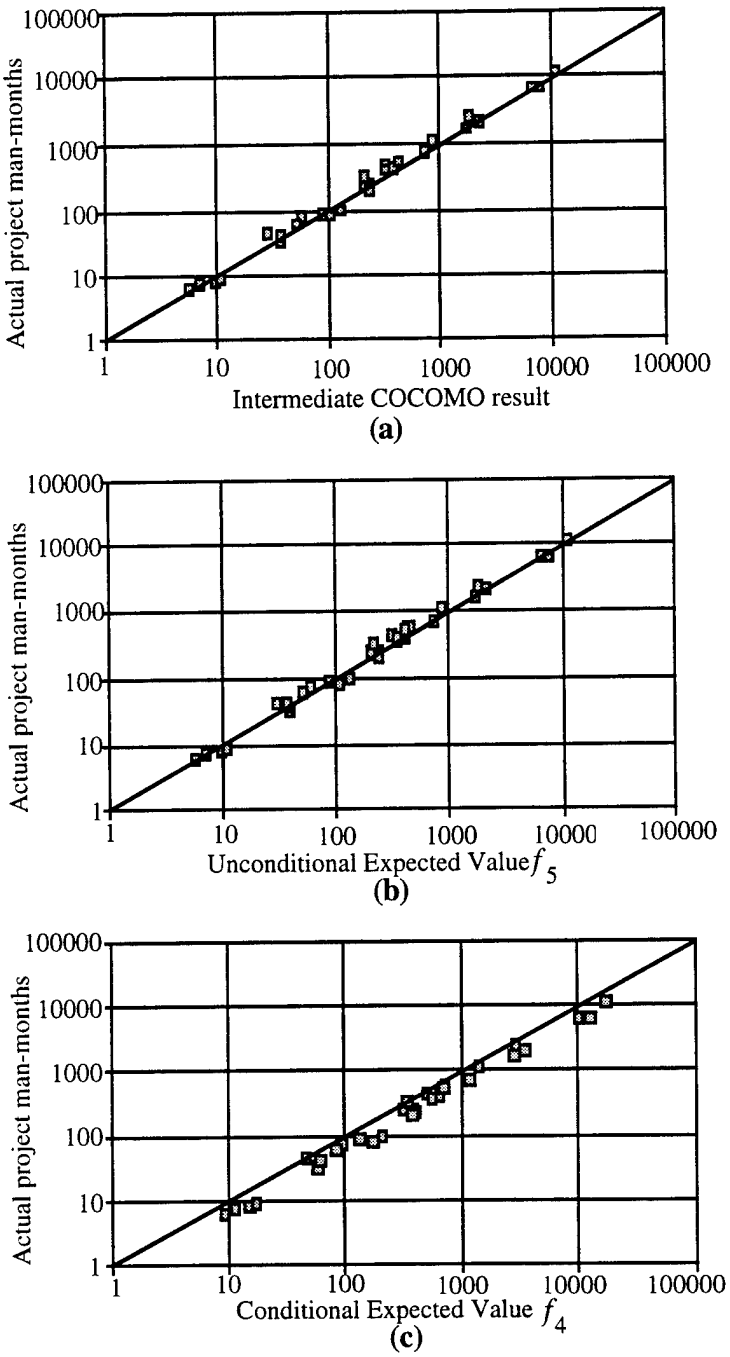


Figure 5.7 Intermediate COCOMO model results: (a) Original forecast versus actuals, (b) Unconditional expected values versus actuals, (c) Conditional expected values versus actuals.

above, but for left-skewed input distributions (indicating an underestimation of KLOC requirements) would even more strongly indicate the relationship of the conditional expectation to the actual results.

5.5.2 Underestimation Comparison Formulation

To test the claim that the conditional expected value is a useful decision making metric when considering the tendency to underestimate project size early in the development life cycle, we consider the scenario in which the parameters of the KLOC input distribution estimates are set at: a = 0.40*(actual KLOC), b = 1.25*(actual KLOC), and c = 0.60*(actual KLOC). Again, all other model parameters are fixed at their original value.

5.5.2.1 Underestimation Comparison: Basic COCOMO. Comparative plots of the Basic COCOMO model values of the unconditional and conditional expected development effort versus actual values are shown in Figure 5.8. In this underestimation scenario, the conditional expected values more closely reflect the actual development effort than do the unconditional expected values. As depicted in Figure 5.8(a), the unconditional expected values generally underestimated the development effort, with most points lying above the principal diagonal. On the other hand, the conditional expectation results are more closely aligned with the actual values (Figure 5.8(b)), showing a great improvement over that of the unconditional expected value results.

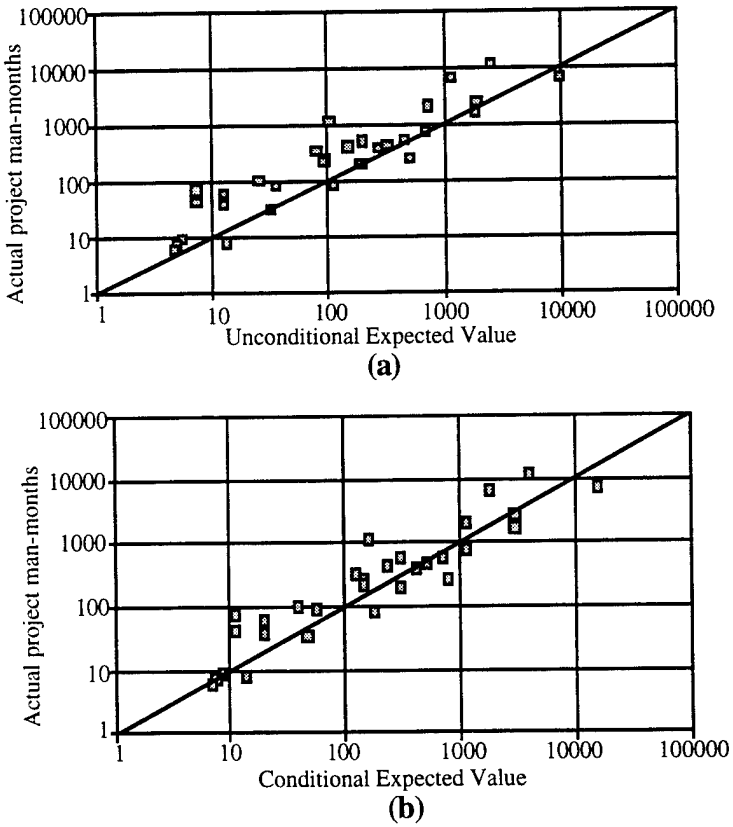


Figure 5.8 Underestimation Scenario -- Basic COCOMO model results: (a) Unconditional expected values versus actuals, (b) Conditional expected values versus actuals.

5.5.2.2 Underestimation Comparison: Intermediate COCOMO. Figure 5.9 depicts the results of the probabilistic assessment of the Intermediate COCOMO model for the underestimation scenario. Again, the improvement of the Intermediate COCOMO model over the Basic model in estimating actual effort is easily noticed. As with the Basic model, the unconditional expected values underestimate the actual development effort (Figure 5.9(a)), but the deviation from actual values is much less than with the Basic model. Figure 5.9(b) shows the remarkable accuracy with which the conditional expected values correspond with actual development effort values.

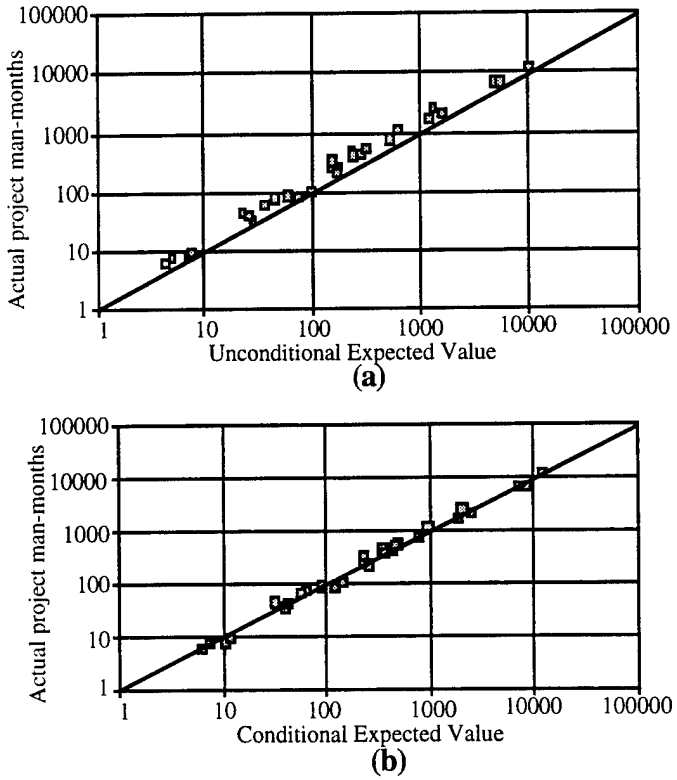


Figure 5.9 Underestimation Scenario Intermediate COCOMO model results: (a) Unconditional expected values versus actuals, (b) Conditional expected values versus actuals.

Histogram plots of the normalized percentage error (Figure 5.10), indicate the increased value of examining the conditional expectation under the circumstances of parameter underestimation. The conditional expected value results are much more closely aligned about the zero point (indicating exact correlation), while the unconditional expected values underestimate the actual development effort.

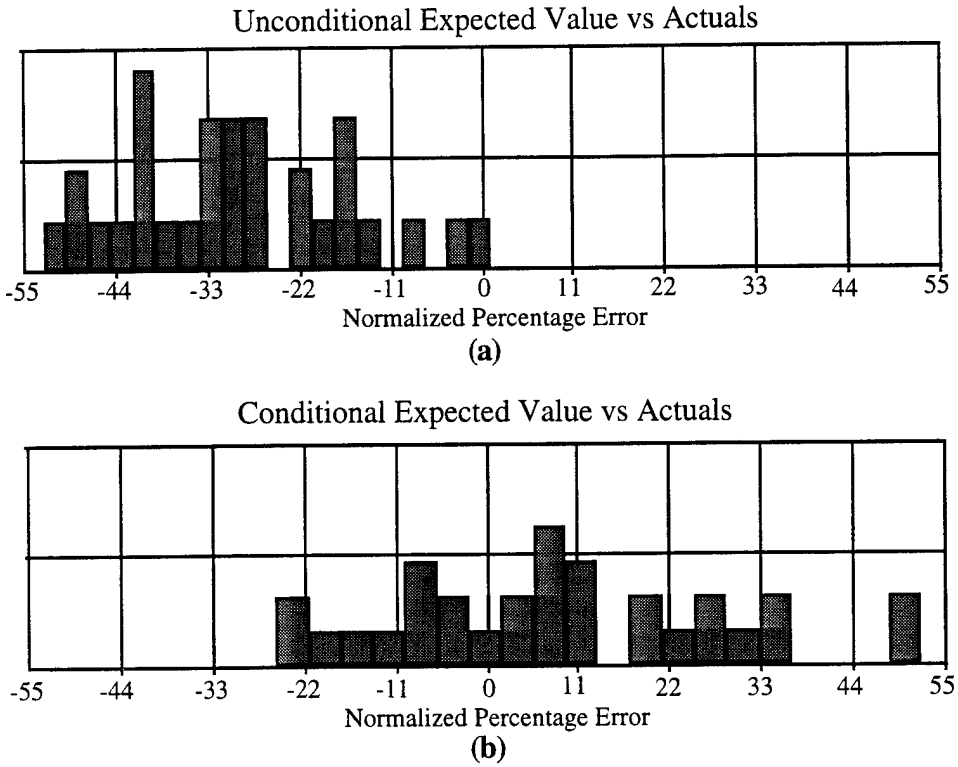


Figure 5.10 Underestimation comparison: Intermediate COCOMO model normalized percentage error: (a) Original forecast versus actuals, (b) Unconditional expected values versus actuals, (c) Conditional expected values versus actuals.

5.6 Chapter Summary

In this chapter, we have developed and demonstrated a probabilistic approach to software estimation. This included both a direct approach and a Monte Carlo simulation approach to software estimation. We also developed a method for determining the conditional and unconditioanl expected values from a Monte Carlo simulation. The probabilistic approach was tested using the original COCOMO data set, and the results were compared to those from the COCOMO.

The added benefit of the probabilistic approach for software estimation, particularly the use of the conditional expected value used to supplement the more traditional unconditional expected value, provides a greater representation of the risks of extreme events associated with a particular policy or alternative. As uncertainty is at its greatest early in the life cycle when model parameter estimates are least certain, the need for supplementing the traditional expected value analysis with that of the conditional expected value is critical for decision making. The conditional expected

value provides not only an understanding of the extreme-event possibilities, but also indicates the potential effects that could be realized from underestimating model parameters.

The results of the comparison with actual development effort underscore the consistency of the probabilistic approach with the original deterministic form of the COCOMO model. This provides a measure of confidence for the software cost estimation community as they employ a probabilistic approach -- the method behaves in a manner consistent with the traditional approach. The results also indicate the importance of using the conditional expected value for analysis, particularly early in the development life cycle when project parameters such as project size are often underestimated.

While the probabilistic approach has been demonstrated for the Basic and Intermediate COCOMO models, the flexibility of this methodology permits its application to the full range of software cost estimation models.

Chapter 6
Dynamic, Multistage Software Estimation

Discrete-time dynamical models that incorporate the probabilistic estimation approach of the previous Chapter are developed for the software estimation problem. These models, with increasingly complex forms of the state and observation equations and expanding probabilistic representation flexibility, provide a range of dynamic modeling formulations for software estimation. Recognizing that practical limitations often exist in data form and availability, these models allow selection of a dynamical model most appropriate for a specific situation. First, a linear-normal dynamical model with closed-form solution is developed. This model provides the context for defining the components of the software cost estimation dynamical model and for describing the interaction of these components. Then, the second model relaxes the linearity and normal distribution restrictions, employing nonlinear state and observation equations derived from the Intermediate COCOMO model.

6.1 Introduction and the Need for Dynamic Software Estimation

Software acquisition is not generally considered a static decision activity. Rather, as captured in the spiral model of software development [Boehm 1988], the process consists of multiple repetitions of primary stages that often extends over a great length of time. Lederer and Prassad [1993] report that in practive, software estimation is most often prepared at the initial project proposal stage; then, with declining frequency, at the requirements, systems analysis, design, and development stages. However, as the software development community continues to move away from the traditional waterfall development process model to the spiral-type models, the demand has increased for cost estimation models that account for the dynamics of changing software requirements and design (and the always-present uncertainty) over multiple time periods. Bell's survey of software development and software acquisition professionals indicate that a vast majority believe a dynamic software estimation model would be most applicable for their estimation requirements [Bell 1995].

At each stage of the acquisition process, decisions are made that affect the events and decision opportunities of subsequent phases. Software estimation is a required activity in

each of the stages of the process. Applying the probabilistic cost estimation method with its multiple objective risk functions, f_i, described in the previous Chapter, constitutes a multiple objective decision problem that is solved over multiple stages. We next give a brief background on dynamic modeling and multiobjective, multistage trade-off analysis. Then, the remainder of the Chapter is a development of dynamical models for software estimation.

6.1.1 Dynamic Modeling -- The Basic Problem

Dynamic modeling is the term applied to methodologies that are concerned with sequential decision problems that involve a dynamic system [Bertsekas 1976], [Reid 1983]. Such systems have an input-output description and system inputs are selected sequentially after observing past outputs. The formulation of optimal control of a dynamic system is very general since the state space, control space, and uncertainty space are arbitrary and may vary from one state to the next. The system may be defined over a finite or infinite state space. The problem is characterized by the fact that the number of stages of the system is finite and fixed, and by the fact that the control law is a function of the current state. (Problems where the termination time is not fixed or where termination is allowed prior to the final time can be reduced to the case of fixed termination time [Bertsekas 1976]).

The discrete-time dynamic system is given by

$$x(k+1)=f(x(k),\ u(k),\ w(k)) \tag{6.1}$$

where $x(k)$ is the state of the system at stage k, $u(k)$ represents the control, or policy implemented at that stage, and $w(k)$ accounts for the random "disturbance" not otherwise captured in the model. The system output associated with each stage is given by

$$y(k) = g(x(k),\ v(k)) \tag{6.2}$$

where $y(k)$ is a cost or other output metric associated with the state of the system, $x(k)$ is the state of the system, and $v(k)$ is another purely random sequence accounting for randomness in the observation process.

Given an initial state $x(0)$, the problem is to find a control policy sequence that minimizes both the sum of all output costs $y(k)$, $k = 1, ..., N$ and the cost associated with the implementation of the control policies $u(k)$, $k = 1, ..., N$.

Figure 6.1 depicts the dynamical model that has been described. The input to each stage includes the state value from the previous stage *x(k)*, a policy input *u(k)*, and the effect of random process disturbances *w(k)*. These are used in Eqs. (6.1) and (6.2) to produce the cost estimate output *y(k)* and to update the state variable *x(k+1)*.

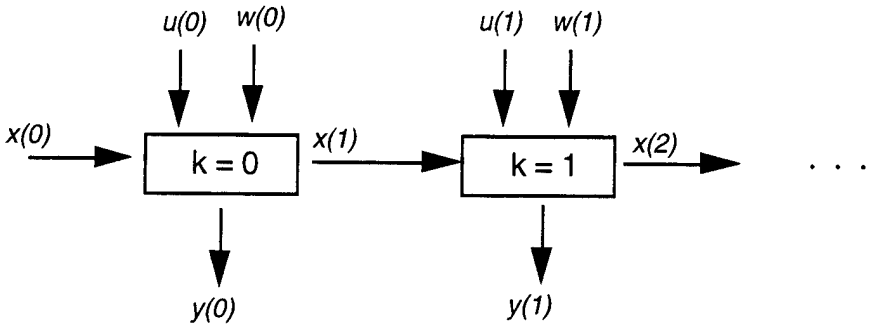


Figure 6.1 Discrete-Time Dynamical Model

6.1.2 Multiobjective, Multistage Tradeoff Analysis

As the objective of the dynamical model is to find a cost-minimizing control policy sequence, the trade-off among project cost versus policy costs must be examined. Gomide and Haimes [1984] developed a theoretical basis for impact analysis in a multiobjective framework. In their multiobjective multistage impact analysis method (MMIAM), the trade-off decision metric is the marginal rate of change of one objective function f_i per unit change in another objective function f_j. Applying the concepts of the MMIAM along with that of the PMRM in a dynamical model introduces the concept of the stage trade-off. The stage trade-off, given by λ_{ij}^{kl}, represents the marginal rate of change of $f_i^k(x,u,k)$ per unit change in $f_j^l(x,u,l)$. These trade-offs provide a measure of the impacts upon levels of the risk objective functions at various stages. Additional discussion concerning full and partial trade-offs is given in [Haimes and Chankong 1979], [Chankong and Haimes 1983], and [Gomide and Haimes 1984].

6.2 Dynamical Modeling for Software Estimation

As the acquisition process progresses through its several stages, the knowledge regarding the project is updated and the uncertainty is (hopefully) reduced. More specifically, the greater the understanding of the software project as a whole, the better one can estimate key

systems characteristics. From this information, appropriate project management policies regarding resource allocation and systems requirements can be made.

Each stage k of the model represents a decision point in the software acquisition process. These include such milestones as the formal milestone decision points of the federal government acquisition process [DoD 1991], and the less formal, yet more frequent, intermediary review points: preliminary design review (PDR), software specification review (SSR), critical design review (CDR), etc.

For the software cost estimation problem, we define the state variable $x(k)$ to be the estimated KLOC required for the intended system. As a state variable, KLOC conveys the overall characterization of the complexity and feasibility of the desired software system. The system output at each stage of the acquisition process, $y(k)$, is the development effort or cost of the software project. The functional form of $y(k)$ may be that of one of the software cost estimation models described earlier.

The estimated KLOC requirement of a software system can be impacted in several ways, most notably from: i) the characteristics or attributes imposed on the system, ii) the resource allocation and acquisition strategy policies, and iii) external factors. Each of these factors are accounted for in the state equation. The performance threshold levels imposed on a system are those metrics required to meet the operational requirements of the user community. Some of these factors are: system reliability requirements, software purpose (functionality), execution or turn-around time, and computational throughput [Boehm 1981], [Sage 1995]. For example, requiring a high degree of system reliability may require greater KLOC for the system. System constraints often increase the complexity of the intended system, further contributing to greater KLOC requirements.

The control policy, $u(k)$, represents the acquisition strategy control and project control decisions that are selected. This includes the type and amount of non-budgetary resources expended for software development. The control policies affect the KLOC requirement for the project and also influence the overall cost of the project. The resource allocation policies considered in this model concern two principal non-budgetary resources, namely personnel and technology. Personnel policy decisions relate to the selection and utilization of personnel with suitable qualifications (highly skilled, skilled, limited knowledge, etc.) and experience. Technological resources include the availability and allocation of specific programming languages and programming tools, the employment of certain programming

practices, and data base and storage resources.

While there are numerous external factors that impact a software system's characteristics, one common external factor is the user community's changing operational requirements. The dynamic world of the user often results in modifications to the originally-specified requirements and functionality of the system. Other external influences that impact the KLOC requirement for the system include political factors, technology advances, and the current status of the software development industry. All these external factors have a possible effect on the system complexity, the estimated KLOC requirements, and the resource allocation policies.

6.3 A Linear Dynamical Software Estimation Model

Having introduced the general form of the state and output equations and having defined the model elements for a software cost estimation context, we develop a dynamical model for software cost estimation. While this initial model assumes a linear relationship among the parameters, it is anticipated that reality will often dictate a more complex formulation. The intent of this initial model, however, is to describe the general dynamics of the estimated size of the intended software system (measured in KLOC), the control policy and system constraints, and the resultant cost output associated with these elements. The initial model also serves as a vehicle for describing the application of dynamical modeling to software acquisition. Having used a linear model to accomplish these purposes, we will relax the linearity requirement in following extensions.

In addition to the model parameters described above, we consider the output of each stage, $y(k)$, to be a vector output as we consider the unconditional as well as the conditional expectation functions associated with the output function. We also introduce a cost function, f_1^k, that accounts for the cost of implementing the chosen control policy at each stage. The problem is to choose a control sequence $\{u(1), u(2), u(3), ..., u(n)\}$ so as to minimize the policy implementation cost as well as the development cost vector.

The dynamics of the system are described by

$$x(k+1) = cx(k) + du(k) + w(k) \tag{6.3}$$

and the output equation for each stage, representing the cost of project development is

given as

$$y(k) = ax(k) + v(k). \tag{6.4}$$

The multiobjective cost estimation problem *for each stage* is stated as follows

$$\begin{aligned} &\text{Minimize:} && \boldsymbol{f}^k = \left\langle f_1^k,\ f_4^k,\ f_5^k \right\rangle \\ &\text{Subject to:} && x(k+1) = cx(k) + du(k) + w(k) \\ & && y(k) = ax(k) + v(k) \end{aligned} \tag{6.5}$$

where

k	represents the discrete stages (decision points) of the system
$x(k)$	is the state of the system, the estimated KLOC input to stage k
$y(k)$	is the calculated effort (cost) output of stage k
$u(k)$	is the resource allocation and acquisition strategy control policy of stage k
$w(k)$	is a random variable accounting for process noise
$v(k)$	is a random variable accounting for observation noise
a	is a cost-per-KLOC multiplier measured in equivalent terms as $y(k)$
c	is the KLOC-adjustment multiplier reflecting system and environment attributes
d	is a KLOC requirements-per-selected policy multiplier
f_4^k	is the conditional expectation of the output variable $y(k)$ at stage k
f_5^k	is the unconditional expectation of the output variable $y(k)$ at stage k
f_1^k	is the cost of implementing control policy $u(k)$.

6.3.1 Solution Approach for the Linear Dynamical Problem

The solution to a deterministic formulation of the problem given by Eq. (6.5), in which the values of all model parameters are known with certainty and the preferred control policy is ascertained, is a straightforward application of multiobjective math programming methods (see [Chankong and Haimes 1983]). In order to introduce the consideration of uncertainty and variance in the model parameters, we apply the probabilistic approach of Leach and Haimes [1987] to describe the model parameters where the disturbances $v(k)$ and $w(k)$ are permitted to be normally distributed, purely random sequences with mean zero and variance σ_v^2 and σ_w^2 respectively (constant for all k).

The selection of normal random variates is based on the knowledge that any linear combination of normal random variables is also a normal random variable [Ross 1989].

Thus, examining Eq. (6.3) we conclude *x(k+1)* is a normally distributed random variable and, therefore, so is *y(k)* by Eq. (6.4). Leach and Haimes [1987] and Romei [Romei et al. 1992] derive exact-form solutions for the unconditional and conditional expected values of normally-distributed functions with the form of Eqs. (6.3) and (6.4). The conditional expectation objective function f_4^k of the normally-distributed *y(k)*, is defined in terms of the mean $\mu(k)$ and variance $\sigma^2(k)$ of the cost distribution. The conditional expectation on the region $[s_k, t_k]$, $s_k < t_k$, is [Leach and Haimes 1987]:

$$f_4^k(u) = \mu(k;u) + \beta_4^k \sigma(k) \tag{6.6}$$

where

$$\beta_4^k = \frac{\int_{s_k'}^{t_k'} \frac{\tau}{\sqrt{2\pi}} e^{-\tau^2/2} d\tau}{\int_{s_k'}^{t_k'} \frac{1}{\sqrt{2\pi}} e^{-\tau^2/2} d\tau} \tag{6.7}$$

$$t_k' = \frac{t_k - \mu(k)}{\sigma(k)}, \quad s_k' = \frac{s_k - \mu(k)}{\sigma(k)}, \tag{6.8}$$

$$\mu(k;u) = \mathrm{E}[y(k)] = \mathrm{E}[ax(k) + v(k)] = a\mathrm{E}[x(k)] + 0 = a\mathrm{E}[x(k)], \tag{6.9}$$

$$\sigma^2(k) = \mathrm{Var}[y(k)]. \tag{6.10}$$

As *y(k)* represents cost, the conditional expectation (6.6) is an objective function to be minimized.

The unconditional expected cost, f_5^k, is the expected value of the output cost function and, using Eq. (6.9), and can be represented as

$$f_5^k = \mathrm{E}[y(k)] = a\mathrm{E}[x(k)]. \tag{6.11}$$

The general solution to Eq. (6.11) can be proven by induction [Haimes and Li 1995], resulting in

$$f_5^k = \mathrm{E}[y(k)] = ac^k x_0 + \sum_{i=0}^{k-1} ac^i du(k-1-i). \tag{6.12}$$

Observe from Eqs. (6.6), (6.7), and (6.10), that the term $\beta_4^k \sigma(k)$ is a factor of k only, and not of the control *u(k)*. Therefore, minimizing the conditional expected value function (6.6) is reduced to minimizing the unconditional expected value:

$$\min_u f_4^k(u) = \min_u \left\{ \mu(k;u) + \beta_4^k \sigma(k) \right\} = \beta_4^k \sigma(k) + \min_u \mu(k;u). \tag{6.13}$$

This implies that minimizing the mean of *y(k)*, i.e., minimizing *μ(k;u)*, should yield the same controls as minimizing f_4^k. Because of this, the trade-offs associated with the conditional and unconditional expectation functions for any given *k* will be equal. Only the levels of the objectives will be different. In other words, the expectation functions at stage *k* are parallel lines.

Using the results of Eq. (6.13) and the fact that the variance is independent of the control, we can consider a deterministic system model that is equivalent to the stochastic one described by Eqs. (6.3) and (6.4) but without the elements of randomness -- for the optimization process all random variables are assigned the value of their mean. Let ^ denote the equivalent variables for the deterministic system, Eqs. (6.3) and (6.4) become:

$$\begin{aligned} &\hat{x}(k+1) = c\hat{x}(k) + d\hat{u}(k) \\ &\hat{y}(k) = a\hat{x}(k), \quad \text{and} \\ &\hat{x}(0) = x_0. \end{aligned} \tag{6.14}$$

Solving Eq. (6.14) for $\hat{y}(k)$ yields the same solution as Eq. (6.12) [Haimes and Li 1995]. Hence the important result:

$$\hat{y}(k) = f_5^k = \mathrm{E}[y(k)] = \mu(k;u). \tag{6.15}$$

An outline of a methodology for solving the multiobjective, multistage problem (6.5), is given by [Leach and Haimes 1987]:

1. Determine the partitioning scheme for each component of damage (cost) for each stage and calculate the values of all β_4^k.
2. Calculate the variance $\sigma^2(k)$ for each stage.
3. Formulate the equivalent deterministic system (6.14).
4. Include the deterministic cost equation $\hat{y}(k)$ with the other objective functions in finding noninferior solutions.
5. The value of $\hat{y}(k)$ is equal to the unconditional expected value. Determine the conditional expected values by Eq. (6.6). Tradeoffs for a given stage are the same for all conditional expected values are equal to the stage tradeoffs calculated for $\hat{y}(k)$.
6. Use a multiobjective decision-making method such as the surrogate

worth tradeoff (SWT) method [Haimes 1980] to find the preferred solution.

6.3.2 Example 6.1 - Policy Evaluation using the Linear Dynamic Software Estimation Model

The following is an example of how the multistage model described in the previous section may be applied. The model is a stochastic, time invariant, linear difference equation representing the relationship between software development management control policies, estimated model size, and project cost. Three stages are considered here, representing original cost and system requirement estimates obtained through a pre-bid conference, which are then updated at decision points early in the Requirements Determination and Design phases of the software acquisition process.

Let $x(k)$, the state variable at stage k representing estimated KLOC, be expressed as a ratio to the initial estimate. The initial state is known with certainty, hence $x(0) = 1$. The control policy $u(k)$ (level of resource allocation) is expressed as a ratio to the nominal level of allocations, just before the beginning of the planning horizon. Implementation of a particular policy is selected as a risk-prevention measure -- reducing the risk of excessive project cost overruns. This value can be considered as incorporating the personnel and product elements of the Intermediate COCOMO model [Boehm 1981], along with acquisition management options such as additional review and study, the hiring of external consultants, requirements for the development of prototype systems, etc.

The cost-per-KLOC constant, a, is fixed at $1 million, an often-quoted figure for mission-critical flight control software [Rifkin 1995]. Let the performance characteristics constant, c, be fixed at $c = 1.44$, representing increasing complexity due to operational demands imposed on the system. This value is obtained by considering the product and computer attributes of the Intermediate COCOMO model [Boehm 1981]. The parameter d, the KLOC-adjustment due to policy selected, is fixed at $d = -0.25$. This value is negative, assuming a modest moderating effect of the application of resources on the otherwise increasing system complexity. Finally, let $w(k)$ represent an external random disturbance with mean zero and variance $\sigma_w^2 = 0.04$. The system's representation is then:

$$x(k+1) = 1.44x(k) - 0.25u(k) + w(k)$$
$$y(k) = x(k)$$
$$x(0) = 1$$

$$\mu_w = 0$$
$$\sigma_w^2 = 0.04$$
$$u(k) \geq 0$$
$$k = 0,1,2,3. \tag{6.16}$$

For this example, the present-value cost function associated with the implementation of a particular policy is given by

$$f_1 = \sum_{k=0}^{n-1} K\left[u(k)-1\right]^2 \left(1+\tfrac{r}{2}\right)^{-2k} \tag{6.17}$$

where $K = \$100*10^3$, $r = 10\%$ is the annual discount rate, and the time period between stages is 6 months. Note that the cost function does not change with time -- the dynamics are incorporated through the present value.

Following the procedure outlined above, we now formulate the deterministic system. The multiobjective optimization that includes the project cost output, Eqs. (6.6) and (6.11), and the control policy implementation cost (6.17) can now be stated as

$$\min_{u(k)} \begin{bmatrix} f_1^0(u(k)) \\ f_i^1(u(k)) \\ f_i^2(u(k)) \\ f_i^3(u(k)) \end{bmatrix}, \quad i = 4,5 \qquad \text{(one } i \text{ at a time).}$$

Using the ε-constraint approach [Chankong and Haimes 1983] to generate the needed Pareto optimal solutions, the problem formulation is given as

$$\min_{u(k)} f_1^0(u(k)$$
$$\text{s.t.} \quad \begin{aligned} f_i^1(u(k)) &\leq \varepsilon_1 \\ f_i^2(u(k)) &\leq \varepsilon_2 \\ f_i^3(u(k)) &\leq \varepsilon_3 \end{aligned} \quad , \; i = 4,5.$$

This leads to forming the Lagrangian function,

$$L(\bullet) = f_1^0 + \lambda_1(f_i^1 - \varepsilon_1) + \lambda_2(f_i^2 - \varepsilon_2) + \lambda_3(f_i^3 - \varepsilon_3), \quad i = 4,5 \tag{6.18}$$

where the Lagrange-multipliers describing the trade-offs between the cost function and risk functions are represented by

$$\lambda_k = \lambda_{1i}^{0k} = -\frac{\partial f_1^0}{\partial f_i^k}. \tag{6.19}$$

To solve the multiobjective optimization problem, we need only generate the unconditional expected cost function, f_5^k, for each stage k = 1, 2, 3 (due to Eqs. (6.13) and (6.15)). Applying Eqs. (6.11), (6.12), and (6.16) produces the following unconditional expectation functions for each stage:

$$\begin{aligned} f_5^1 &= \mathrm{E}[y(1)] = a\mathrm{E}[x(1)] \\ &= a\mathrm{E}[cx(0)] + du(0) + w(0)] = ac\mathrm{E}[x(0)] + ad\mathrm{E}[u(0)] + \mathrm{E}[w(0)] \\ &= (1)(1.44)(1) + (1)(-0.25)u(0) + 0 \\ &= 1.44 - 0.25u(0); \end{aligned}$$

$$\begin{aligned} f_5^2 &= \mathrm{E}[y(2)] \\ &= (1.44)^2 - 0.25u(1) - (1.44)(0.25)u(0) \\ &= 2.074 - 0.36u(0) - 0.25u(1); \end{aligned}$$

$$\begin{aligned} f_5^3 &= \mathrm{E}[y(3)] \\ &= (1.44)^3 - 0.25u(2) - (1.44)(0.25)u(1) - (1.44)^2(0.25)u(0) \\ &= 2.986 - 0.25u(2) - 0.36u(1) - 0.5184u(0). \end{aligned}$$

Substituting the above three results and that of Eq. (6.17) into Eq. (6.18), the Lagrangian is now

$$\begin{aligned} \mathrm{L}(\bullet) = {} & [100(u(0) - 1)^2 + 90.70(u(1) - 1)^2 + 82.27(u(2) - 1)^2] \\ & + \lambda_1[1.44 - 0.25u(0)] \\ & + \lambda_2[2.074 - 0.36u(0) - 0.25u(1)] \\ & + \lambda_3[2.986 - 0.25u(2) - 0.36u(1) - 0.5184u(0)]. \end{aligned} \tag{6.20}$$

Taking the derivatives of Eq. (6.20) with respect to the controls at $u(2)$, $u(1)$, and $u(0)$ and applying first-order stationary conditions, we determine the trade-off values (6.19). Table 6.1 gives three possible noninferior solutions. For each solution, the Table gives the values of the control variables, the value and the levels of the risk functions, and the trade-off values between the cost and risk functions.

Policy A represents no change in resource allocation over the planning horizon. Because there is no additional application of resources, no policy implementation costs are incurred. However, the conditional and unconditional expected project costs become increasingly

Table 6.1 Noninferior Policies for Software Acquisition

Stage	Risk Function	Trade-offs
Policy A[a]		
$k = 1$	$f_4^1 = 1.467$	
	$f_5^1 = 1.190$	$\lambda_{14}^{01} = \lambda_{15}^{01} = 0$
$k = 2$	$f_4^2 = 1.741$	
	$f_5^2 = 1.464$	$\lambda_{14}^{02} = \lambda_{15}^{02} = 0$
$k = 3$	$f_4^3 = 2.135$	
	$f_5^3 = 1.858$	$\lambda_{14}^{03} = \lambda_{15}^{03} = 0$
Policy B[b]		
$k = 1$	$f_4^1 = 1.442$	
	$f_5^1 = 1.165$	$\lambda_{14}^{01} = \lambda_{15}^{01} = 863.62$
$k = 2$	$f_4^2 = 1.642$	
	$f_5^2 = 1.365$	$\lambda_{14}^{02} = \lambda_{15}^{02} = 181.40$
$k = 3$	$f_4^3 = 1.868$	
	$f_5^3 = 1.591$	$\lambda_{14}^{03} = \lambda_{15}^{03} = 329.08$
Policy C[c]		
$k = 1$	$f_4^1 = 1.342$	
	$f_5^1 = 1.065$	$\lambda_{14}^{01} = \lambda_{15}^{01} = 804.84$
$k = 2$	$f_4^2 = 1.436$	
	$f_5^2 = 1.159$	$\lambda_{14}^{02} = \lambda_{15}^{02} = 362.80$
$k = 3$	$f_4^3 = 1.570$	
	$f_5^3 = 1.293$	$\lambda_{14}^{03} = \lambda_{15}^{03} = 329.08$

[a] Control variables: $u(0) = 1$, $u(1) = 1$, $u(2) = 1$; cost, $f_1^0 = 0$ ($\$10^3$).

[b] Control variables: $u(0) = 1.10$, $u(1) = 1.25$, $u(2) = 1.50$; cost, $f_1^0 = 27.24$ ($\$10^3$).

[c] Control variables: $u(0) = 1.5$, $u(1) = 1.5$, $u(2) = 1.5$; cost, $f_1^0 = 68.24$ ($\$10^3$).

worse over time. By the third stage, the expected value of project cost has become 1.858, with the extreme-event conditional expected value at 2.135. The trade-offs between all risk functions and cost are zero (due to no implementation costs), so that small improvements in the risk functions can be made at little additional cost. Because the trade-offs are zero, this is an improper noninferior solution [Chankong and Haimes 1983].

Policy B is a policy of gradual increase in personnel and technological resources allocated for project development. The expected project cost increases less dramatically over the time period, and the conditional and unconditional expected values indicate less risk than policy A. The lower project costs over those of policy A are achieved with relatively low policy implementation costs. This will be demonstrated graphically.

Policy C represents an immediate increase in resource allocation. The result is a significant decrease in expected project cost, with the expected value rising to only 1.293 by the third stage. Of the three solutions in Table 1, policy C is the one of lowest risk, but it is also the most expensive. The trade-offs are also much larger for this policy, indicating that it becomes increasingly expensive to gain additional improvement in the risk functions.

Selection of the most preferred of the three noninferior policies must be made by the decision maker, taking into account his/her personal preferences in the trade-offs between the cost function and the risk functions. Formal methods such as the surrogate worth tradeoff (SWT) method [Haimes 1980] are appropriate. Analysis of the impact that control policies have on later- stage decision making options must also be taken into account.

To demonstrate why impact analysis is so useful in a problem such as this one, suppose the multiobjective problem was solved only one stage at a time. The cost associated with resource allocation policy control (6.17) in the first stage, denoted by f_1^1, is

$$f_1^1 = 100[u(0) - 1]^2.$$

Figure 6.2 shows the set of noninferior solutions when the first-stage costs f_1^1 and the expected damage at the first stage f_5^1 are the only objectives considered. The points corresponding to the policies A, B, and C are indicated on the curve. Considering only the first-stage objectives, the selection of policy C over the other alternative policies would appear desirable; the initial $25,000 policy implementation cost produces an expected $125,000 project cost reduction over the project's life cycle.

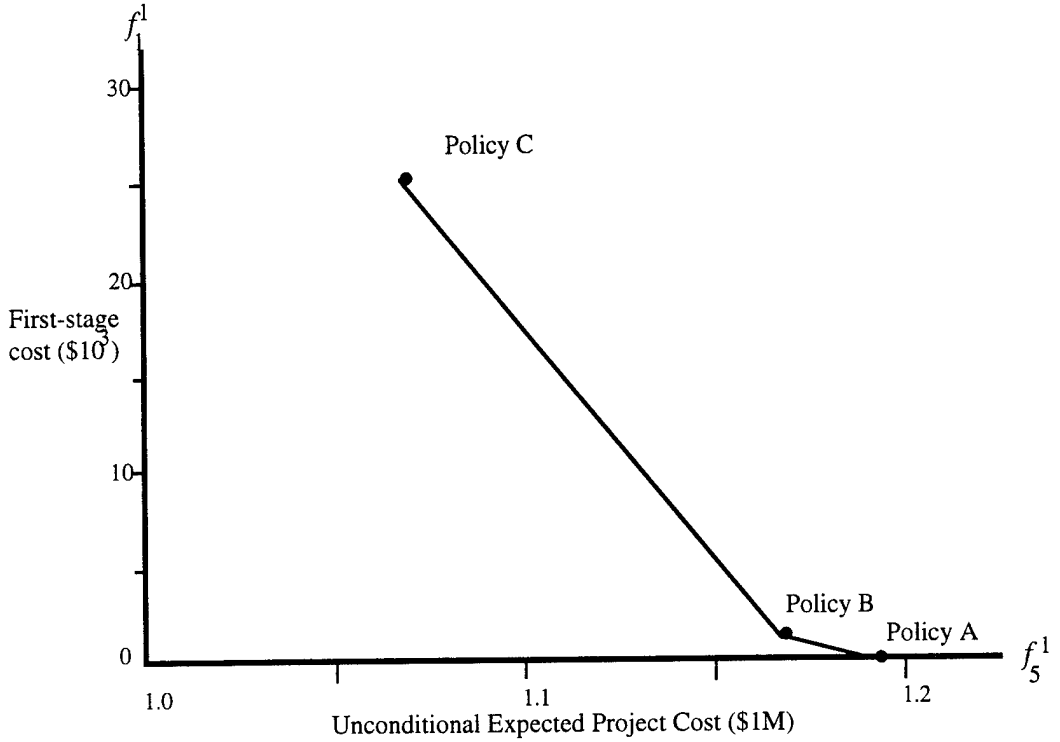


Figure 6.2 Noninferior solution set considering only first-stage objectives

Consider now the second stage, with the cumulative control costs, denoted by f_1^2, given by

$$f_1^2 = 100[u(0) - 1]^2 + 90.70[u(1) - 1]^2.$$

Depending on which policy was implemented in the first stage, three different noninferior solution sets are possible in the second stage, as shown in Figure 6.3. Each curve is labeled with its associated first-stage policy. The way in which the first-stage policy affects the second-stage (and subsequent-stage) decision making is what makes impact analysis desirable. Li and Haimes [1987] [1988] show that there is a family of such noninferior solution sets, where each curve depends on the chosen policy of the previous stage. The envelope of this family of curves engulfs all the noninferior solutions of each stage, thereby defining the noninferior frontier for the multistage problem. Additional decision-making information can be provided by plotting the conditional expectation curves for each alternative policy. Trade-offs are then made in terms of both expectation values.

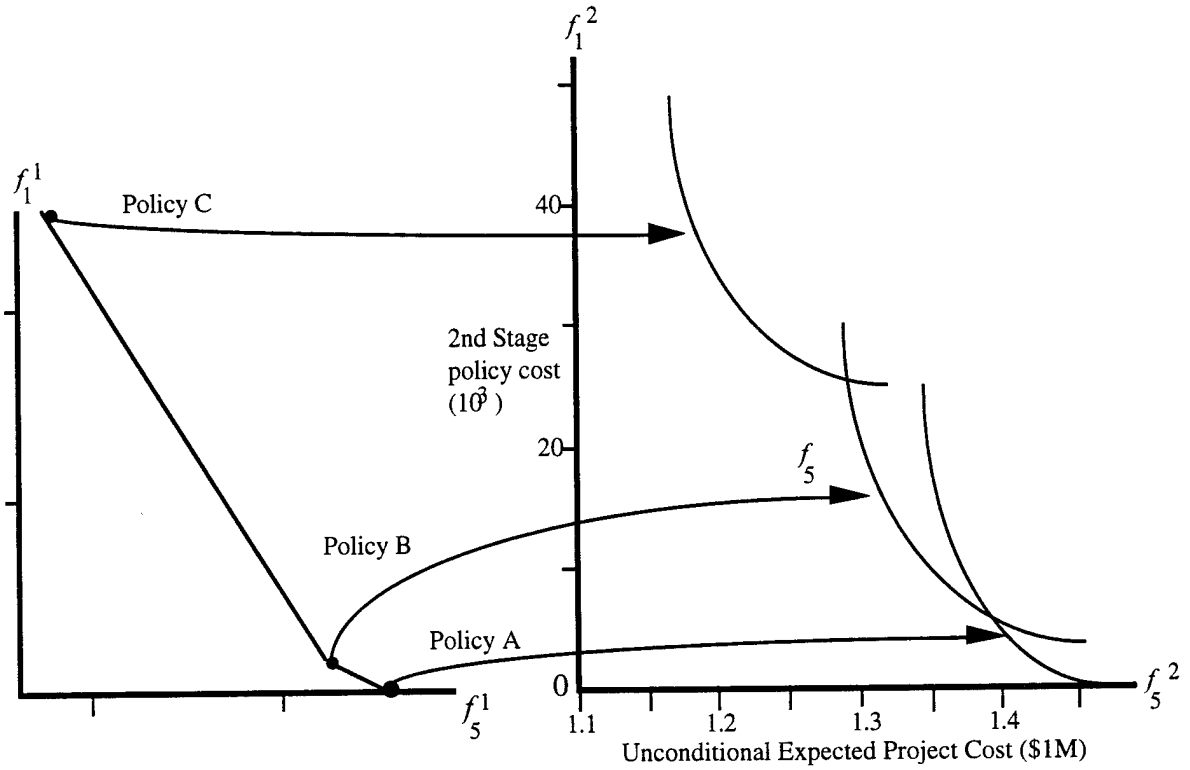


Figure 6.3 Impact analysis at the second stage

6.3.3 Observations

The linear, multistage software estimation model has provided a necessary framework for understanding and analyzing the interactions of the software cost estimation parameters. The closed-form solution enabled an analytical description of the dynamics of the software cost estimation model parameters. The example problem demonstrated the benefits to decision making by using this approach -- both in terms of the importance of impact analysis and multiobjective tradeoff analysis. This model sets the stage for the development of a multistage software cost estimation model that is more closely associated with existing methods.

6.4 A Nonlinear Multistage Software Estimation Model

While the linear dynamical formulation provides a general representation of the interactions of the state, control, and external factors in software cost estimation, the linear relationships imposed in the model do not necessarily represent actual parameter interactions. It is

desirable to relax some of the conditions of the above model and develop a multistage dynamical model that is more closely aligned with existing estimation methods. Unlike Eq. (6.5), this new model constitutes a nonlinear formulation. Such a formulation precludes the expectation for an analytic closed-form solution. This is not a deterrent, however, due to the availability of powerful computer software. We formulate this more-realistic nonlinear dynamical software cost estimation model and use a Monte Carlo simulation approach for its analysis. The form of the state and output equations, as well as an example problem that demonstrates the application of this revised model, are based on the Intermediate COCOMO model.

Recall that the KLOC-based software development effort models, such as the COCOMO model, produce both nominal and modified development effort values, where the modified value is based on the existence of certain project attributes. The nominal man-months of development effort equation has the nonlinear form [Boehm 1981]

$$MM_{NOM} = a(KLOC)^b \tag{6.21}$$

which written in the notation of Eqs. (6.1) and (6.2) gives

$$y(k)_{NOM} = a(x(k))^b. \tag{6.22}$$

The Intermediate COCOMO model's effort adjustment factor (EAF) then revises the nominal effort result according to the available resources, project requirements, and environmental attributes (similar to the earlier definitions of $u(k)$ and c), resulting in the development effort requirement

$$MM = (EAF)MM_{NOM} = (EAF)[a(KLOC)^b] \tag{6.23a}$$

or

$$y(k) = (EAF)[a(x(k))^b]. \tag{6.23b}$$

EAF is defined as [Boehm 1981]:

$$\text{EAF} = \prod_{i=1}^{15} e_i \tag{6.24}$$

where e_i is an effort multiplier associated with a particular cost multiplier attribute.

The adjusted effort (6.23b) may also be viewed as the result of a revision in the KLOC requirements. In other words, the existence of a requirement for a certain program attribute, or the implementation of a certain control policy, impacts the KLOC requirement

for the system (e.g., increasing the system reliability requirement may increase the KLOC). The relationship between the effect of system attributes and control policy on KLOC and Boehm's EAF can be found by solving

$$a\left[u(k)c(k)(KLOC)\right]^b = (EAF)\left[a(KLOC)^b\right] \tag{6.25}$$

for the product $u(k)c(k)$, where $c(k)$ represents system and environment attributes at stage k and $u(k)$, as defined earlier, represents the control policy of stage k. Solving Eq. (6.25) for $u(k)c(k)$ leads to

$$\left[u(k)c(k)(KLOC)\right]^b = (EAF)(KLOC)^b$$

$$u(k)c(k)(KLOC) = (EAF)^{1/b}(KLOC)$$

$$u(k)c(k) = (EAF)^{1/b}. \tag{6.26}$$

Hence, the combined adjustment factor to KLOC due to the system characteristics and the selected control policy at stage k is quantified as the b^{th} root of Boehm's effort adjustment factor. The system output at each stage of the process (6.23b), defined as the projected development effort of the intended system $y(k)$, can now be represented as

$$y(k) = a\left[\left(c(k)u(k) + v(k)\right)x(k)\right]^b \tag{6.27}$$

where $v(k)$ is a random sequence accounting for the influence of external factors on the project's development. Equation (6.27) reflects the combined effects that the state of the system (the estimated KLOC), the control policy, specific system attributes, and external forces have on the project's eventual development effort.

The state update equation, the sequential revision of the estimated KLOC, is given by

$$x(k+1) = \left[c(k)u(k) + w(k)\right]x(k) \tag{6.28}$$

where the random sequence $w(k)$ accounts for the influence of external factors, $c(k)$ reflects system and environment attributes, and $u(k)$ represents the resource allocation and acquisition strategy policies employed at that stage. Equations (6.27) and (6.28) provide the mathematical description of this nonlinear discrete-time system for software cost

estimation, paralleling Eqs. (6.4) and (6.3) of the linear model.

The formulation for this nonlinear, dynamical model for software cost estimation can now be written. As with the previous model, the objective functions to be optimized at each stage include the conditional and unconditional expected values of project development effort and the implementation cost associated with the selected control policy. The model consists of an equality (6.27) that relates the current state *x(k)* and the current policy *u(k)* to obtain the output value *y(k)*. Also, it includes a difference equation (6.28) that relates the current state *x(k)* and the control policy *u(k)* to obtain the next state *x(k+1)*. The multiobjective problem *for each stage* is:

$$\begin{aligned} &\text{Minimize:} && \boldsymbol{f}^k = \left\langle f_1^k,\ f_4^k,\ f_5^k \right\rangle \\ &\text{Subject to:} && x(k+1) = \left[c(k)u(k) + w(k)\right]x(k) \\ & && y(k) = a\left[\left(c(k)u(k) + v(k)\right)x(k)\right]^b \end{aligned} \tag{6.29}$$

where:

k	represents the discrete stages (decision points) of the system
$x(k)$	is the state of the system, the estimated KLOC input to stage k
$y(k)$	is the calculated effort (cost) output of stage k
$u(k)$	is the resource allocation and acquisition strategy control policy of stage k
$c(k)$	is a composite adjustment multiplier to the required KLOC reflecting system and environment attributes at stage k
$v(k),\ w(k)$	are composite random variables that account for external disturbances
$a,\ b$	are parameters depicting system characteristics, as with the COCOMO models [Boehm 1981]
f_4^k	is the conditional expectation of development effort $y(k)$ at stage k
f_5^k	is the unconditional expectation of development effort $y(k)$ at stage k
f_1^k	is the cost of implementing control policy $u(k)$.

6.4.1 Solution Approach for the Nonlinear Dynamical Problem

Having relaxed the linearity requirements of the previous formulation, we also no longer require the random variables to be normally distributed -- allowing representation by any suitable probability distribution. In addition to *v(k)* and *w(k)*, we also permit probabilistic representations of *x(k)*, *c(k)*, and *u(k)*.

A common practice in the solution of problems such as Eq. (6.29) is to set *v(k)* and *w(k)* to their expected values in order to permit a more focused exploration of the impact and interaction of the state and control variables [Leach and Haimes 1987]. Application of this technique, however, overlooks the fact that these variables are often the most critical, as they are the unpredictable changes of policies and requirements during the software lifecycle and that removing them from consideration in the model may impact analysis of the model's results.

Considering the nonlinear relationships among model parameters in this new formulation, and that these parameters may be represented by a variety of probability distributions, we can no longer assume a closed-form solution for the cost output distribution, *y(k)*, or its conditional expected values f_4^k and f_5^k. We are interested in computing E[*y*(X)] and E[*y*(X)|X>*x*], where the notation, X (actually X(*k*)), indicates that the estimated KLOC is now treated as a random variable. Since it is not analytically possible to compute a closed-form solution of the expected values, we apply a simulation approach.

To approximate E[*y*(X)] for each stage, we generate a random value $X^{(1)} = x$ from the density function of X and then compute $Y^{(1)} = y(X^{(1)})$. We next generate a second random value (independent of the first) $X^{(2)}$ and compute $Y^{(2)} = y(X^{(2)})$. This continues until *n*, a fixed number of independent and identically distributed random variables $Y^{(i)} = y(X^{(i)})$, $i = 1, \ldots, n$ have been generated. By the strong law of large numbers [Ross 1989] we know that

$$\lim_{n\to\infty} \frac{Y^{(1)} + \cdots + Y^{(n)}}{n} = E[Y] = E[y(X(k))] = f_5^k. \tag{6.30}$$

By Eq. (6.30) we can use the average of the generated $Y^{(i)}$s as an estimate for E[*y*(X(*k*))], the unconditional expected value. To approximate the conditional expected values, f_4^k, we form the sub-set of outcomes Y_β whose members are those outcomes that exceed the partitioning value β associated with the predetermined α value. Simply stated,

$$Y_\beta = \left\{ Y^{(i)} \middle| Y^{(i)} \geq \beta \right\}. \tag{6.31}$$

The average of the members of Y_β is the average of all outcomes that exceed a particular damage level -- precisely the definition of the conditional expected value. Hence, given *m* elements of the set Y_β (for *m* sufficiently large), an approximation to the conditional expected value of the development effort (cost) is given by

$$\frac{Y_\beta^{(1)} + \cdots + Y_\beta^{(m)}}{m} \cong E\left[Y_\beta\right] = E\left[y(X(k)) \middle| y(X(k)) \geq \beta\right] = f_4^k. \qquad (6.32)$$

Monte Carlo simulation of the present-value cost function of implementing a particular control policy f_1^k is conducted in a similar manner. The unconditional and conditional expected control policy costs are generated from the simulation outcomes.

While the initial density function for KLOC, $X(0) = f(x_0)$, may have a pre-specified form, subsequent updates of the KLOC probability distribution cannot be obtained analytically. Therefore, a Monte Carlo simulation of the stage-wise update for the state equation (6.28) is similar to the procedure described for the effort and policy cost equations. The n outcomes, $X^{(i)}(k+1)$, provide the data for probability distributions of the simulated outcome results that are the state input distributions for the following stages.

An outline of a methodology for solving this new multiobjective, multistage problem is:

1. Determine the partitioning scheme for each component of damage (cost) for each stage.
2. Solve the probabilistic problem using Monte Carlo simulation to produce a distribution for the cost output at each stage and to update the state distribution.
3. Determine the conditional and unconditional expected values of the simulation cost distribution results. Use localized variations to estimate the tradeoffs for a given stage.
4. Use a multiobjective decision-making method such as the surrogate worth tradeoff (SWT) method to find the preferred solution.

6.4.2 Example 6.2 - Policy Analysis using the Nonlinear Dynamic Software Estimation Model

This example problem demonstrates how the nonlinear dynamical model (6.29) is applied and solved. A natural extension of a probabilistic software cost estimation, the model is a stochastic, nonlinear difference equation representing the relationship between software development resource allocation control, estimated model size, and project cost. The example employs probabilistic representation of model parameters and demonstrates the power of modern analytic support tools to solve complex mathematical problems.

We adapt for this example, the problem described in [Boehm 1981] concerning the development of a semidetached software product. We account for the inherent uncertainty associated with the early stages of a software acquisition effort by quantifying the initial estimate of KLOC requirements, *x(0)*, as a triangular probability density function. This distribution is derived from considering three values for *x(0)*: a low, most likely, and high estimate (Table 6.2). As the project is intended to be a 32-KLOC product, this value is taken as the Most Likely. A slightly lower, Low value and a High value one and one-half times that of the Most Likely are assumed.

Table 6.2 Triangular distribution parameters for initial KLOC estimate, *x(0)*

	Low	Most Likely	High
x(0)	28	32	48

For this example problem we use a present-value control policy cost function that reflects the direct relationship of *u(k)* and its implementation cost; greater allocation of resources increases the policy costs. The complexity of the system within which the control policy is implemented also affects the cost of that policy. The control policy implementation cost, as a function of both the policy and the environmental attributes within which that policy is implemented is given by:

$$f_1^0 = \sum_{k=0}^{n-1} K\left[c(k)(2-u(k))\right]\left(1+\tfrac{r}{2}\right)^{-2k}. \tag{6.33}$$

We set $K = \$1*10^6$, $r = 10\%$, and the time period between each of the n stages is 6 months. In this new formulation, the possibility is open for parameters K and r to also be represented by probability distributions.

The effect of system and environmental attributes *c(k)* is quantified by considering the effort adjustment factor of the Intermediate COCOMO model's product and computer complexity attributes. The ratings and scores of the system attributes of the first stage are listed in Table 6.3. Each attribute is rated on a scale from Very Low to Extra High, and each rating has an associated KLOC adjustment factor. Nominal ratings have an adjustment factor of 1.0. Note that a high system attribute rating translates to an increased KLOC requirement. While each attribute's score in Table 6.3 could be represented by a probability distribution, we will instead use single values for this example. Furthermore, for this example, these values are held constant for all k. Using the data from Table 6.3 in Eq. (6.24), the contribution to the EAF due to system attributes is 1.438. By Eq. (26),

Table 6.3 System Complexity Attribute KLOC adjustment factors

Attribute	Rating	Adjustment Factor (e)
Reliability	Nominal	1.00
Data Base Size	Low	0.94
Complexity	Very High	1.30
Execution Time	High	1.11
Storage	High	1.06
Virtual Machine Volatility	Nominal	1.00
Turnaround Time	Nominal	1.00

$c(k)$ = 1.383. This value reflects the increased system complexity, execution time, and storage requirements reported at this stage, resulting in an increasing trend for KLOC requirements.

Quantification of the control policy, $u(k)$, is governed by the resource allocation elements of the Intermediate COCOMO model's personnel and project attributes. As above, each element is rated on a scale from Very Low to Extra High, with each rating having an associated quantitative KLOC adjustment factor. The resource allocation adjustment factors impact KLOC requirements and resource allocation costs in opposite directions -- a low rating (implying limited allocation of qualified resources) results in an increased KLOC requirement, but a lower implementation cost. This explains the differences in the functional form of $u(k)$ in Eqs. (6.29) and (6.33). Table 6.4 lists the control policy ratings and scores for the resource allocation policy for one stage. Similar ratings and scores are determined for each control policy at each stage in the process. Using the values from Table 6.4 in Eq. (6.24) and applying Eq. (6.26), the adjustment to KLOC due to the control policy is 0.832. The employment of highly-qualified analysts and programmers, along with advanced programming practices leads to efficient, accurate programming that requires less KLOC.

Table 6.4 Resource allocation control policy KLOC adjustment factors (from [Boehm 1981])

Resource	Allocation	Adjustment Factor (e)
Analyst Capability	High	0.86
Applications Experience	Nominal	1.00
Programmer Capability	High	0.86
Virtual Machine Experience	Low	1.10
Programming Language	Nominal	1.00
Programming Practices	High	0.91
Software Tools	Low	1.10
Development Schedule	Nominal	1.00

The constants, a and b, are assigned the values from the Intermediate COCOMO model that correspond to the development environment. For a semidetached product, $a = 3.0$ and $b = 1.12$ [Boehm 1981]. The two random variables that account for external disturbances, $v(k)$ and $w(k)$, are assumed to be normally distributed with mean zero and variance $\sigma_v^2 = \sigma_w^2 = 0.04$.

6.4.2.1 Model Verification. In order to verify the formulation of this model, we compare the solution of its deterministic formulation with the results from [Boehm 1981]. Setting each probabilistic parameter to its most likely value gives $x(0) = 32$ KLOC, $v(0) = 0$, and $w(0) = 0$. The initial evaluation of the nominal effort for this example problem is found by Eq. (6.22):

$$y(k)_{NOM} = a(x(k))^b = (3.0)(32)^{1.12} = 146 \text{ man-months (MM)}, k = 0.$$

Using Eq. (6.26), the impact on the estimated KLOC requirement due to the observed system attributes $c(0)$ and chosen policy $u(0)$ is

$$c(0)u(0) = (1.171)^{1/1.12} = 1.151.$$

The effect of the system attributes and control policy in revising the estimated KLOC is given by Eq. (6.28):

$$x(1) = c(0)u(0)x(0) = 36.832 \text{ KLOC}.$$

Hence, the adjusted development effort output from the first stage as a function of the estimated KLOC and the control policy is found by Eq. (6.27):

$$y(1) = y(x(1)) = (3.0)[c(0)u(0)x(0)]^{1.12} = 170.33 \text{ MM}.$$

This result is very close to the rounded solution of 171 MM given in [Boehm 1981]. Similar, near-exact solutions were produced for additional example problems also from [Boehm 1981].

6.4.2.2 Probabilistic Evaluation. We now extend the application of the nonlinear dynamical model beyond single-stage estimation, where the results of stage 1 provide the input for stage 2. In each subsequent stage, a new control policy is implemented, reflecting the decision maker's resource allocation strategy for the development of the intended system. This, along with a revised observation of system and environment attributes and a

new estimate of the KLOC requirement, is used to revise the development effort output in light of the updated knowledge.

Allowing each parameter to assume its probabilistic form, the Monte Carlo simulation approach described above is used for solving the development effort output equation (6.27), state equation (6.28), and control policy cost equation (6.33).

Similar to the linear example problem, consider three representative, noninferior control policies (Table 6.5). Policy A represents a nominal resource allocation, policy B is a gradual increase in the amount and quality of allocated resources, and policy C is an immediate and sustained increase in resource allocation. These policies may be considered optimistic, since even policy A has no resource category with a worse-than-nominal allocation. The partitioning value for the conditional expected value was set at $\alpha = 0.9$, thus f_4^k reflects the 1-in-10 extreme event.

Table 6.5 Noninferior Policies for Nonlinear Multistage Software Acquisition Example

	Policy A[a]	**Policy B**[b]	**Policy C**[c]
Stage	Risk Function	Risk Function	Risk Function
$k = 1$	$f_4^1 = 535.66$	$f_4^1 = 445.41$	$f_4^1 = 199.25$
	$f_5^1 = 344.54$	$f_5^1 = 276.78$	$f_5^1 = 122.17$
$k = 2$	$f_4^2 = 833.68$	$f_4^2 = 493.35$	$f_4^2 = 172.97$
	$f_5^2 = 506.54$	$f_5^2 = 285.32$	$f_5^2 = 106.16$
$k = 3$	$f_4^3 = 1353.76$	$f_4^3 = 423.25$	$f_4^3 = 151.71$
	$f_5^3 = 714.86$	$f_5^3 = 182.71$	$f_5^3 = 80.43$

[a] Control variables: $u(0) = 1$, $u(1) = 1$, $u(2) = 1$; cost, $f_1^0 = 1.35$ ($\$10^6$).

[b] Control variables: $u(0) = 0.9$, $u(1) = 0.8$, $u(2) = 0.6$; cost, $f_1^0 = 1.51$ ($\$10^6$).

[c] Control variables: $u(0) = 0.6$, $u(1) = 0.6$, $u(2) = 0.6$; cost, $f_1^0 = 2.09$ ($\$10^6$).

Since policy A has no application of resources beyond the nominal level, its implementation cost is the lowest of the three policies considered. However, the conditional and unconditional expected development effort results become increasingly worse over time. By the third stage, the expected development effort is 715 MM, with the extreme-event conditional expected value a terrifying 1354 MM. This additional risk information regarding potential catastrophic events is the essential advantages of the probabilistic approach. The information regarding the range of possible outcomes associated with each

alternative benefits the decision maker in the process of selecting the most desired alternative. In this example, the conditional expected value information identifies that Policy A has a 1-in-10 potential of an almost doubling its the required development effort.

Risk mitigation policies B and C reflect the effects of the additional allocation of resources -- the required development effort decreases. The gradual approach of policy B shows marked improvement over policy A: the expected development effort increases less dramatically over the time period, even beginning to decrease by stage 3; the conditional and unconditional expected values indicate less risk than policy A. Policy C shows dramatic improvement over the other two options -- both at the first stage, and over the long run. With a development effort expected value of only 80.43 after the third stage, policy C is the one of lowest risk. It is also the most expensive -- over half again the cost of policy A and a third again the cost of policy B. .

A plot of the first stage conditional and unconditional expected development effort values for each policy against its implementation cost is shown in Figure 6.4. By including the conditional expected value in the graph, the decision maker can graphically comprehend the additional information to consider when trading-off among the control policy options. One observes the relative little difference between the conditional and unconditional expected development efforts for policy C, compared with the larger difference for the other two policies. Such risk reduction comes at a cost, however. Policy C is 38% more expensive than that of policy B, but reduces the expected development effort to less than half of that required under policy B. Policy B is 12% more expensive than policy A, yet produces a nearly 25% reduction in the expected development effort requirement. Conducting similar analysis of the extreme event values, we see an even more-pronounced difference in the policies.

Again, analysis of the impact of early-stage decisions on later-stage decision opportunities (as in Section 6.3.2) is readily calculated.

6.5 Chapter Summary

In this Chapter we have extended the traditional application of software cost estimation methods by developing multistage, dynamical software cost estimation models. Both the linear and nonlinear dynamical models showed promise as multistage software cost

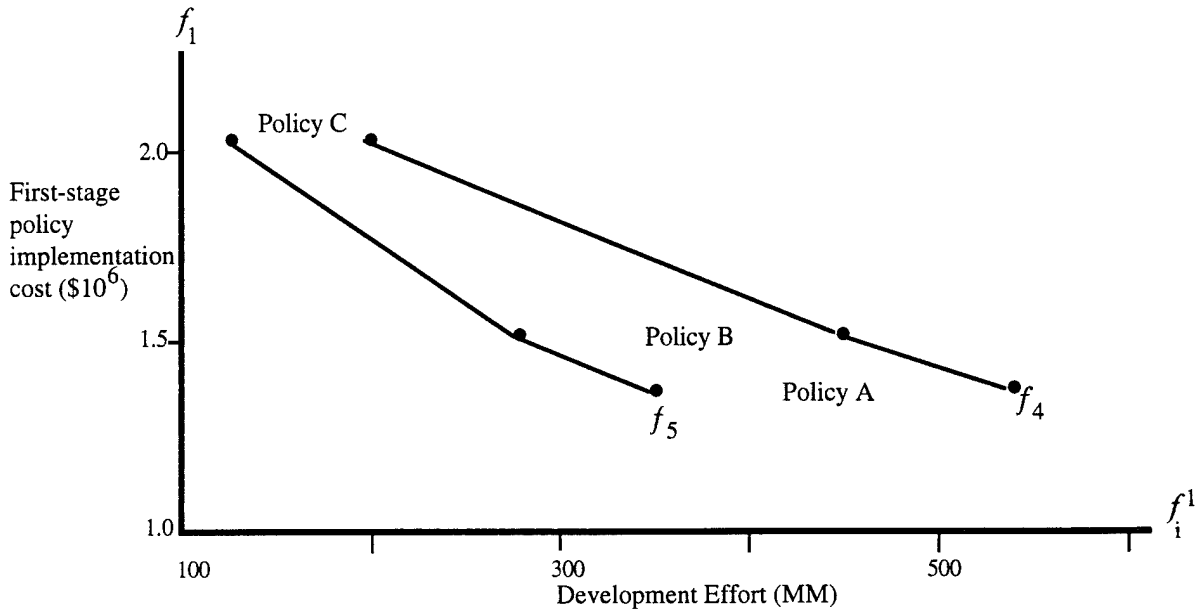


Figure 6.4 Noninferior solution set considering only the first stage objectives

estimation tools. Although it does not provide for an analytically closed-form solution, the availability of Monte Carlo simulation software makes the nonlinear dynamical model practical and desirable. The nonlinear model offers the greatest opportunity for realistic extension of existing, static software cost estimation models (as Example 6.2 demonstrated for the COCOMO model).

As the software development community continues to move away from the traditional waterfall development models to repetitive, spiral-type models, software cost estimation methods must be responsive to this new development paradigm. No longer a single time-period activity, methodologies for software cost estimation must provide updated estimates by considering the system characteristics, policies, and requirements of a changing environment. One overriding characteristic of this environment, particularly in the early stages of the development life cycle, is the uncertainty regarding the desired software system. To this end, a probabilistic approach that explicitly accounts for parameter variability is required.

The dynamical models developed in this Chapter account for the need to update software cost estimates due to the dynamics of changing requirements, improved system design information, and various resource allocation policies associated with the early stages of the software development lifecycle. Incorporating a probabilistic extension of traditional

software cost estimation methods, the models utilize the conditional expected value as an additional decision-making metric. Stage-wise updating of software cost estimates gives the decision maker greater understanding of anticipated project costs and development effort requirements, as well as information concerning the expected impact of various control policy options in reducing project risk.

Chapter 7
The HHM Framework for Dynamic Software Estimation Updating and for Multiobjective Decision Making Coordination

7.1 Introduction

The probabilistic approach to software estimation of Chapter 5 added explicit consideration of the uncertainty and risk associated with software acquisition endeavors. Then the dynamic formulation of the software estimation process developed in Chapter 6 extended software estimation to a multi-stage, repeated process that parallels modern development paradigms and allows analysis of the impact of current decisions on future opportunities. In this Chapter we demonstrate the extensions of HHM in addressing two remaining software acquisition management issues:

- Actual project effort and schedule are rarely exact duplicates of their estimates. Of concern is how to explain the deviation between estimated and actual values, and then in light of the actual results, how to improve the estimate of the remaining development effort and schedule.

- Software project management policy options do not fall entirely in the domain of the customer nor entirely in the domain of the contractor, yet both parties are affected by each other's decisions. At issue is how to coordinate and resolve the competing issues, objectives, and decision opportunities of these participant communities for the benefit of all.

Hierarchical holographic modeling (HHM) is extended to address these two issues. First, we demonstrate a dynamic updating framework that can be implemented within the dynamic software estimation model of Chapter 6 to provide revised project effort and schedule estimates throughout the live cycle. The investigative framework of HHM -- initially applied for risk identification -- is well-suited for providing on-going understanding of the causes for differences between actual software project progress and the estimated effort and schedule requirements. This understanding is a key element for determining the appropriate actions within the dynamic software estimation updating process.

Second, with revised estimates of project effort and schedule, management control policies can be selected so as to best meet stated objectives. The decision authority concerning these policies, however, does not entirely rest with any one participant community, but is divided among the groups. A hierarchical coordination decision-making scheme is employed to help resolve decision-making conflicts and trade-offs associated with the software project participant communities. Each community's multiobjective decision-making problem includes unique, as well as overlapping objectives, decision variables, and model parameters. HHM gives the structure to assist in achieving mutually acceptable solutions.

7.2 Dynamic Software Estimation Updating

After the initial software estimation has been conducted, the contract awarded, and the formal design and development work started, the need for software cost and schedule estimation has not ended. Evaluating and revising these estimates are a critical project management function.

At the project milestones associated with the life cycle phases, actual effort and development schedule data can be collected. With these data, an important question is asked, "Is the project on track?" Stated a bit differently, "Given that the project has used x MM of effort and taken y months to get to this point, will it still be completed according to the original effort and schedule estimates? If not, what caused the deviation from the original estimate, and what is a new estimate for the effort and schedule?"

Project milestone charts (Figure 7.1) are a tool commonly used for recording and comparing actual effort and schedule results against estimated projections [Rozum 1992]. Unfortunately, these charts don't provide the analytic means for projecting future results or analyzing effects of deviations from previous estimates to answer the above questions. In this section we demonstrate a dynamic software estimation updating methodology to sequentially revise effort and schedule estimates and use the HHM as an aid in identifying contributing factors that may explain any deviation from the expected.

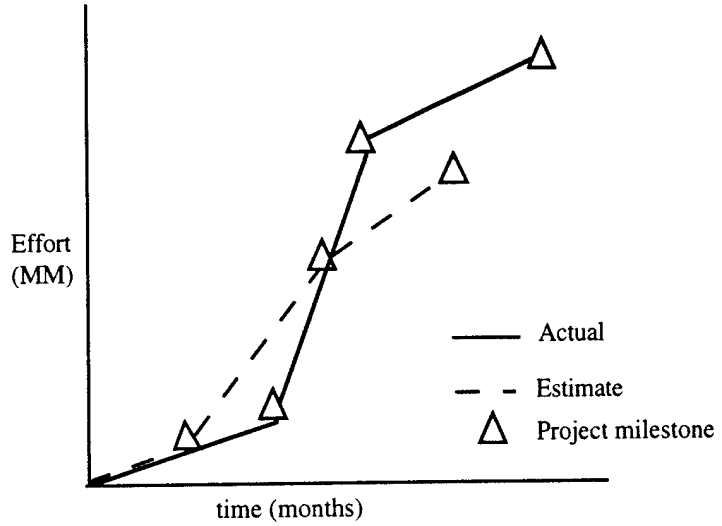


Figure 7.1 Sample project milestone chart

7.2.1 Dynamic Software Estimation Updating Process

Dynamic software estimation updating, as used in this section, refers to the "on-line" revisions of software estimates using the models developed in the previous two chapters. The process involves collecting actual project effort and schedule data and comparing these with previous estimates [Kitchenham and Walker 1989]. The estimation model is then adjusted based on the actual values, and an updated estimate for the remaining effort and schedule is made.

A critical element to updating the estimation model is being able to account for the cause of deviation from previous estimates. We ask, "Of the model parameters and variables, what do we believe? What has changed from the original estimation?" There will often be many possible causes for deviations from estimates, and for each cause there may be several different types of corrective action [Dorflinger and Basili 1985]. Appropriately changing model parameters or variable values to reflect current knowledge leads to an improved estimate.

The three-step process of dynamic software estimation, depicted graphically in Figure 7.2, is:

1. Baseline Estimate. Apply a software estimation methodology, with estimates of project size, environment, and other model parameters, to produce a baseline estimate of project effort and schedule.

2. Actual Observation. Collect known effort and schedule information, along with actual size, environment, and other parameter values, at appropriate project milestone points.
3. Dynamic Estimate. Recalibrate the variable values or equation parameters of the estimation model to produce a revised estimate of the remaining effort and schedule to complete the project. This revised estimate becomes the new baseline estimate.

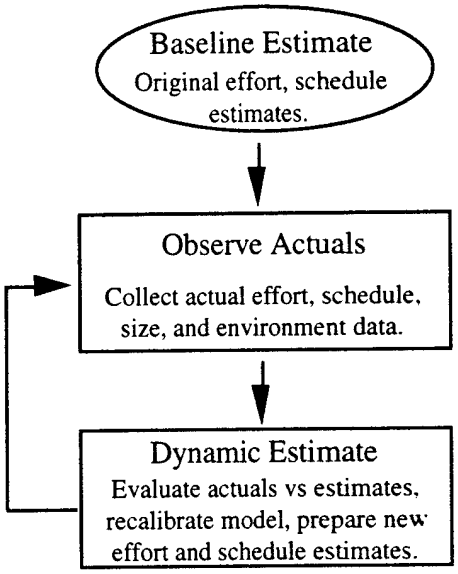


Figure 7.2 Dynamic Software Estimation Updating Process

Step 1 consists of the activities discussed in Chapters 5 and 6 -- establish model parameter and variable values, use an estimation model to determine project effort and schedule, and make policy decisions based on the expected value as well as the conditional expected value.

Step 2 is the simple step of collecting project progress data. While not all data are collected monthly, data collection at project milestones is generally available. These milestones may include the formal event activities [DoD 1991]: system design review (SDR), software specification review (SSR), preliminary design review (PDR), critical design review (CDR), test readiness review (TRR), functional configuration audit (FCA), and physical configuration audit (PCA). These may also include interim milestone events and other less-formal review points.

Step 3 is the actual dynamic estimation activity. This first includes using the HHM to conduct an analysis of the factors that contributed to the project's deviation from the

estimated effort and schedule. Then, the estimation model is calibrated to reflect the observed values by appropriately updating model parameters and variable values according to the results of the HHM analysis. Finally, the calibrated model is used to project a new estimate of project effort and schedule.

7.2.2 Dynamic Software Estimation Updating Methodology

The general methodology adopted for the dynamic software estimation updating problem is from [Kile 1995] and [Kitchenham and Walker 1989]. Let us first consider the effort and schedule equations of either the original static form of the Intermediate COCOMO model or the dynamic form of the model that was developed in Chapter 7 (Table 7.1). In both forms of the model, parameters represent the influence of environmental factors on the development effort requirement, the size (representing complexity) of the project, and the variable relationships.

Table 7.1 Original and Dynamic Intermediate COCOMO equations

Equation	Original Model	Dynamic Model
Effort	$MM = a(EAF)(KLOC)^b$	$y(k) = a\left[\left(c(k)u(k) + v(k)\right)x(k)\right]^b$
Schedule	$t_D = c(MM)^d$	$t_D(k) = c(y(k))^d$

For the software estimation updating activity, we initially utilize an HHM framework with two sub-visions: environment and size (Figure 7.3). The extension of this simplification can be achieved by subsequently adding details to the HHM structure until we have the HHM described in Chapter 3. The environment decomposition represents those elements associated with the system attribute adjustment factors of Chapter 6: system complexity, storage, programming language, software tools, etc.

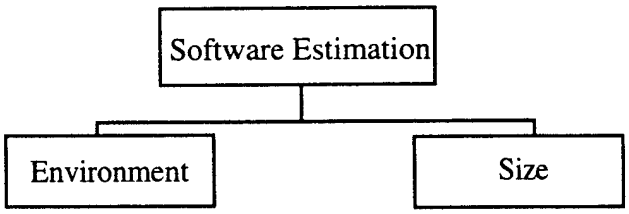


Figure 7.3 Dynamic Software Estimation HHM (Initial Application)

Again, considering two visions (environment and size), if the actual effort or schedule deviate from the estimate, at least one of the following must be true:

- the size estimate is incorrect,
- the environment is specified incorrectly,
- the model's equations require calibration, or
- the stated control policy wasn't implemented (e.g., the program wasn't staffed according to the model).

7.2.2.1 Recalibration Strategy. Depending on which of the above factors is (are) assumed to be the problem, appropriate changes to the model and its elements can be made. Determining the appropriate course of action is facilitated through the use of decision diagrams (Figure 7.4). If one assumes that the stated control policy was, in fact, the implemented policy, then one uses the HHM to evaluate the relative confidence of the size and environment estimates. If the size estimate is deemed valid, then the problem must be with the specification of the environment and the model is recalibrated for a new environment value. On the other hand, if the problem is assumed to be with the size estimate, then recalibrating the model for a new size value is necessary. If both estimates are deemed accurate, or both are viewed as equally inaccurate, then the model's parameters are recalibrated to reflect actual values.

Once the initial problem element has been identified, then appropriate action is taken depending on the specific nature of the actual effort and schedule results versus the estimates. Different adjustments are made for underestimation than are made for overestimation.

7.2.2.2 Recalibration via Environment Specification. Making appropriate changes to a model parameter depends on the nature of the difference between estimated versus actual effort and schedule (depicted in Figure 7.5). If the estimate is greater than the actual effort, then there has been more progress made than was expected (the tasks of the particular milestone were accomplished with less expenditure of effort than was anticipated). Therefore, the estimation model is recalibrated by improving the environment parameter so that the effort estimate is reduced to correspond with the actual value.

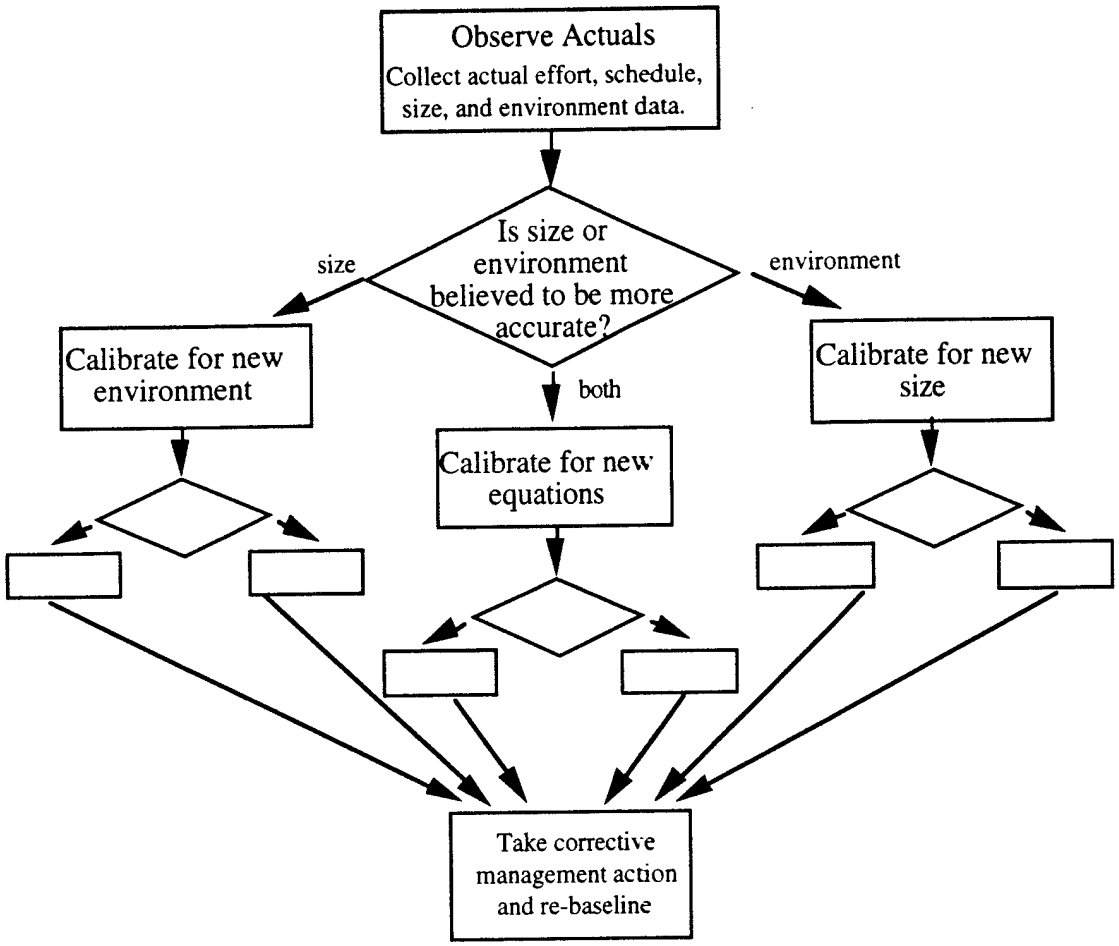


Figure 7.4 Software estimation recalibration strategy (sub-level decision details are described in following figures) adapted from [Kile 1995]

Conversely, if the estimate is less than the actual effort, then the original estimate was overly optimistic and the project is doing worse than was anticipated. The recalibration action is to degrade the environment parameter until the model's estimate increases to the actual value. If the estimated and actual effort values are the same, then no recalibration action needs to be made. Once the model has been appropriately recalibrated, it is used to estimate the project's remaining effort and schedule requirements. This estimate is the new baseline estimate.

7.2.2.3 Recalibration via Size Specification. The recalibration process that modifies the size estimate is very similar to that for the environment (Figure 7.6). Considering only the effect of size, if the estimated effort is greater than the actual value, then the original size estimate was too large. Recalibration is accomplished by decreasing the size parameter so that the effort estimate is reduced to correspond with the actual value.

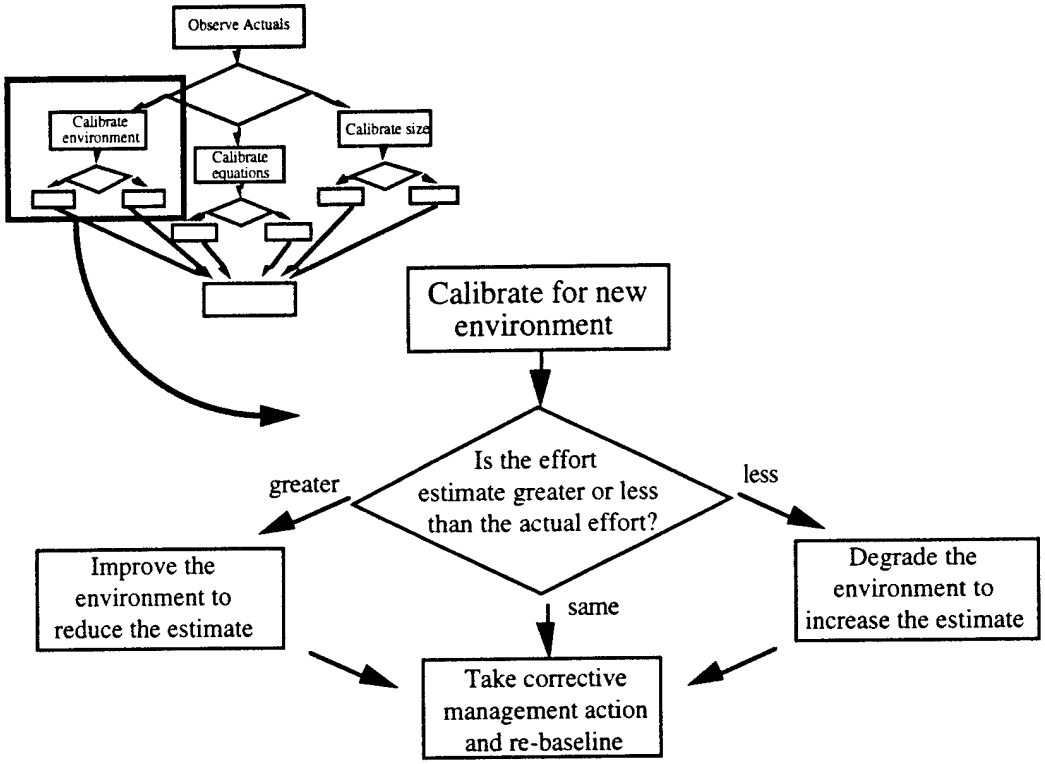


Figure 7.5 Recalibration via environment specification

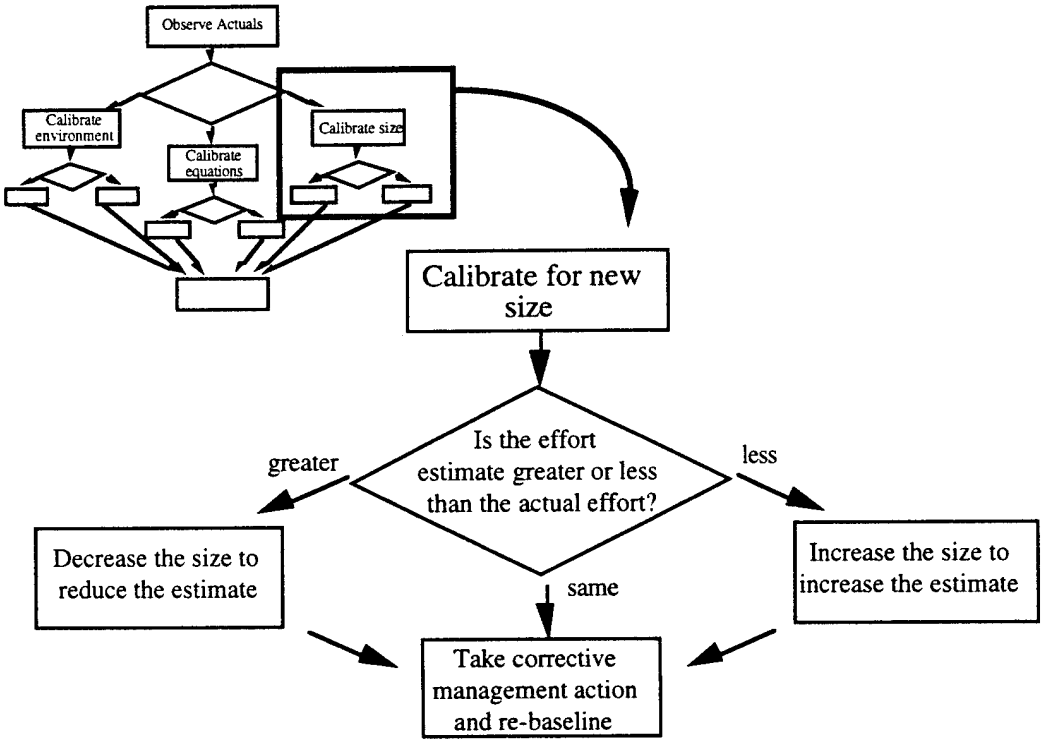


Figure 7.6 Recalibration via size specification

If the estimated effort is less than the actual, then the size was originally underestimated and must be increased to increase the effort estimate to match the actual value. Once the model has been recalibrated, it is used to determine the new baseline estimate of the project's remaining effort and schedule requirements.

7.2.2.4 Recalibration via Model Parameters. Referring to Figure 7.4, if both the size and environment parameters are equally believed (or equally disbelieved), then recalibration of the estimation model is accomplished by modifying the other parameters of the model -- namely the coefficients (e.g., the <*a,c,*> vector of the Intermediate COCOMO model). This can be done in one of three ways: i) change the effort equation coefficient only, ii) solve for the schedule coefficient only, or iii) solve for both the effort and schedule coefficients simultaneously. In either process, the coefficient value is recalibrated so that the actual effort and/or schedule is produced. Either process will affect both of the model equations -- adjusting effort impacts the schedule due to the compression effect, etc.

7.2.3 Example 7.1 - Dynamic Software Estimation Updating

To demonstrate the dynamic software estimation updating approach, consider the development of a 50-KLOC, embedded-mode software system. For this example, we use the original Intermediate COCOMO formulation; the approach for the dynamic formulation would be similar. The model parameter values are listed in Table 7.2. With EAF = 1.00, the original management control policies assume nominal personnel (capability and experience) and nominal environment influences (programming languages, development tools, system complexity, etc.).

Table 7.2 Dynamic software estimation updating example -- initial model values

KLOC =	50
Mode =	Embedded, hence the COCOMO parameters (from Table 3.3)
	a = 2.80
	b = 1.20
	c = 2.50
	d = 0.32
EAF =	1.00

The original, baseline estimate of development effort and schedule is found using Eqs. (3.7) and (3.2)

$$MM = 2.8(1.0)(50)^{1.20} = 306.14 \text{ man-months}$$

and

$$t_D = 2.5(306.14)^{0.32} = 15.6 \text{ months.}$$

Distribution of the estimates by life cycle phase is given in Table 7.3. Recall that the effort and schedule equations are for the development phases of the life cycle. Total project resource requirements through development must also include the plans and requirements phase requirements.

Table 7.3 Software estimation updating example - original estimates

Life Cycle Phase (% of total effort, schedule)	Effort (man-months)	Schedule (months)
Plans & Requirements (8,32)	24.48	5.12
Development		
Design (18, 34)	55.08	5.44
Detailed Design (26,19)	79.56	3.00
Code & Test (28, 21)	85.68	3.40
Integration & Test (28, 26)	85.68	4.16
Total Development	306.00	16.00

Consider that the initial design for the project has been completed and that the preliminary design review (PDR) has been conducted. The original estimate for the required effort to reach PDR was 79.56 MM (24.48 + 55.08) and the schedule estimate to PDR was 10.56 months (5.12 + 5.44). Assume the actual effort expended in reaching PDR was 90 MM, accomplished over a 9-month period. Hence, more effort was utilized to reach PDR than was estimated, but over a shortened schedule. We will now demonstrate the three approaches for accomplishing the software estimation updating in light of this new information.

7.2.3.1 Example 7.1 (cont.) - Accurate size, recalibrate via environment. Assume that the HHM investigation concluded that the size estimate is accurate, while the environment specification is suspect. To recalibrate the environment element, first find the total development schedule that has a distribution of 9.0 months for the first two phases:

$$.66t_D = 9.0 \Rightarrow t_D = 13.64 \text{ months.}$$

Next, determine the total development effort that distributes 90 MM to the first two phases:

$$.26MM = 90 \Rightarrow MM = 346.15 \text{ man - months.}$$

As the development effort estimate was less than the actual effort expended through PDR, degrade the environment factor by increasing the EAF until the appropriate development effort value is given:

$$346.15 = 2.8(EAF)(50)^{1.20} \Rightarrow EAF = 1.131$$

The updated project development effort and schedule distribution is given in Table 7.4

Table 7.4 Software estimation updating example - revised estimates

Life Cycle Phase	Effort (man-months)	Schedule (months)
Plans & Requirements	27.69	4.36
Development		
Design	62.31	4.64
Detailed Design	90.00	2.56
Code & Test	96.92	2.89
Integration & Test	96.92	3.55
Total Development	346.15	13.64

The new development effort and schedule estimates are used to update the effort and time to complete development of the project, which includes the development phase resource requirements as well as that of the earlier phases. We can now update the project's effort requirement 346.15 + 27.69 = 373.84 man-months, and the schedule estimate to 13.64 + 4.36 = 18 months. A plot of the original estimates, actual values, and revised estimates is shown in Figure 7.7. We see that the new estimate calls for more development effort, but has the project being completed in a shorter time than was originally anticipated.

7.2.3.2 Example 7.1 (cont.) - Accurate environment, recalibrate via size. Assume, now, that the environment parameter is correct and that the size parameter must be recalibrated . Following the steps of the above section, find the revised development schedule and effort that correspond with the actual schedule and effort

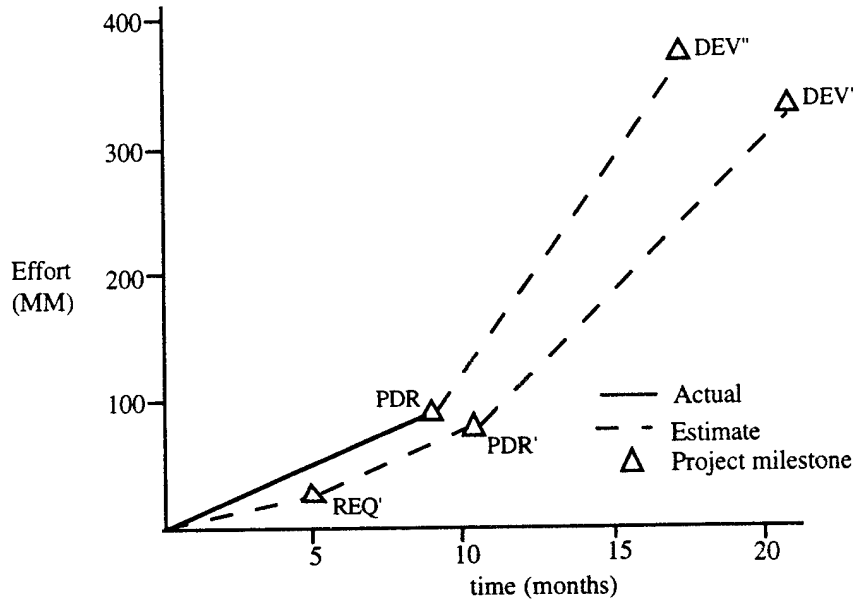


Figure 7.7 Project milestone chart with revised estimates
(DEV' indicates the original estimate of development effort and schedule, DEV" indicates the revised estimate)

expended through the current phase. Then, adjust the KLOC until the appropriate total development effort is given:

$$346.15 = 2.8(1.0)KLOC)^{1.20} \Rightarrow KLOC = 55.40.$$

7.2.3.3 Example 7.1 (cont.) - Accurate size and environment, recalibrate via model parameters. This option is used when both the size and environment specifications are equally believed, or equally disbelieved. Demonstrating the last of the three options for this approach, we first recalibrate the effort coefficient until the total development effort corresponds to the revised projection:

$$346.15 = a(1.0)(50)^{1.20} \Rightarrow a = 3.166$$

Then, recalibrate the schedule coefficient to correspond the total schedule that corresponds with that observed through the current stage:

$$13.64 = c\left(346.15\right)^{0.32} \Rightarrow c = 2.10$$

These new parameters are then used for future estimates, until more-current observations are made.

7.2.4 Software Estimation Updating Summary

The software estimation updating methodology demonstrated in the above example is repeated at each project milestone and at other points in the development process to provide an update on the overall progress and expected resource requirements for the project. The methodology is flexible, easily applied, and provides the needed updated information regarding project progress in terms of expected completion. The provision for expanding the environment and size elements to include additional factors adds increased awareness and robustness to the project management effort.

7.3 Software Acquisition's Multiobjective, Multi-Decision Maker Decision-Making Coordination

As introduced in the dynamic software estimation model of Chapter 6, software management policy decision making is conducted on a recurring basis throughout the life cycle. Updating the software estimation dynamically, and projecting the latest estimate of a project's required effort and schedule, provides the opportunity for updating and adjusting management policy options. These management control options, however, do not fall within any one participant community's domain of control. Rather, each group has its own decision making options, and each is affected by the other's decisions. HHM provides the hierarchical framework for exploring the competing issues, objectives, and decision opportunities for the various participant communities to assist in resolving a solution that is mutually-acceptable to all parties.

7.3.1 Hierarchical Decision Problems

In a classical decision making problem, a decision maker has to select an alternative among those which are acceptable or feasible and that choice is made according to his preferences. In hierarchical decision problems, some of the elements of the decision problem are actually in systems which are related to the strategies adopted by *other* decision makers. In that case, the way resources are allocated is no longer characterized entirely by restrictive constraints or requirements to satisfy, but are characterized as the solutions of *other*

decision making problems, related to the preference structure of the *other* decision makers involved. This leads to a model with a hierarchical structure of decision, with different decision makers possibly located at different levels of decision.

Hierarchical problems involving multiple decision makers have been considered by several authors, traditionally from either the optimization theory point of view (e.g. [Chankong and Haimes 1983], [Haimes et al. 1990], [Bard and Moore 1990], [White 1982], [Bialas and Karwan 1982], [Nijkamp and Reitveld 1981], or from the game theory point of view (e.g., [von Neumann and Morgenstern 1944], [Blackwell and Girshick 1954], [Luce and Raiffa 1957], [Chen and Cruz 1972]). For the software acquisition context, however, the zero-sum decision rule and similar such constructs of a game theory approach are not desirable. Also, many traditional hierarchical optimization methods (e.g., [Installe 1994], [Haimes et al. 1990], [Tarvainen and Haimes 1982]) do not directly apply, for in software acquisition there does not exist an ultimate higher level decision maker -- no higher decision-making authority to whom the user, customer, and contractor all report, and who is directly concerned with all the objectives, etc.

Contrary to the traditional discussion and approach of hierarchical methods with higher-level coordination, we are interested in synchronizing the solutions of the multiple visions (e.g., the user/customer and the contractor) so that ultimately we have an acquisition process not marred by cost overrun and time delay.

The intent is to use the holistic visions of the HHM to provide understanding and accounting for the objectives and constraints of the different decompositions and find a process or mechanism that would bring the necessary collaboration together. With the absence of a higher-level decision maker to dictate a compromise solution between the competing visions, the approach could be considered for application in several contexts:

i) Direct negotiation. Having knowledge of the other decision maker's objectives, data, and possible responses to environment changes, is most important to negotiation [Neirenberg 1978], [Fisher 1981], [Raiffa, 1982]. With each participant aware of the other's problem and model, each is better prepared for negotiations.

ii) Independent analysis. Each participant can solve the problem separately with full knowledge of the other's model (although without full knowledge of the actual

decision variable values of the other decision maker). The differences in each decision maker's solutions can be openly negotiated to a mutually-agreeable level

iii) Iterative coordination. A bi-element, or bi-decomposition iterative approach in which one party sets the level of overlapping variables, the other responds by solving their problem, and the first then re-adjusts the original solution [Installe 1994], [Haimes et al. 1990], [Stackelberg 1952].

7.3.2 Hierarchical Decision Problem Formulation

For this chapter, we consider the existence of two decision decompositions in interaction, denoted respectively as the "α" decomposition and the "β" decomposition. These terms connote a parallel level of decision interaction, versus the traditional "higher" and "lower" decompositions that connote increasing authority. In the α decomposition, the best alternative is selected according to the α decision maker's preference structure, to the set of all feasible alternatives, and to the reaction of the β decomposition decision maker with respect to the decision. In the β decomposition, the decision maker reacts to the α decomposition decision according to his preference structures and according to the feasible alternatives. Hence a bi-element hierarchical decision making problem can be defined using the following mathematical model:

Find $\boldsymbol{x}^{\alpha}$ the best compromise with respect to $\boldsymbol{f}^{\alpha}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha}))$
such that $\boldsymbol{x}^{\alpha} \in X^{\alpha}$ and $\mathbf{g}^{\alpha}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})) \leq 0$

where $\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$ is the solution of the following problem:

Given $\boldsymbol{x}^{\alpha}$, find $\boldsymbol{x}^{\beta}$ the best compromise with respect to $\boldsymbol{f}^{\beta}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta})$
such that $\boldsymbol{x}^{\beta} \in X^{\beta}$ and $\mathbf{g}^{\beta}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta}) \leq 0$ (7.1)

where

$\boldsymbol{x}^{\alpha}$	is the α decomposition decision variables
$\boldsymbol{x}^{\beta}$	is the β decomposition decision variables
$\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$	stands for the reactions of the β decomposition decision makers, given $\boldsymbol{x}^{\alpha}$
$\boldsymbol{f}^{\alpha}$	is the α decomposition objectives; vector valued function of $\boldsymbol{x}^{\alpha}$ and $\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$
$\mathbf{g}^{\alpha}$	is the α decomposition constraints
X^{α}	is the α decomposition definition set
$\boldsymbol{f}^{\beta}$	is the β decomposition objectives

$\mathbf{g}^{\beta}$ is the β decomposition constraints

X^{β} is the β decomposition definition set.

Simplifications of the form of the objectives and constraints (e.g., linear, convex, mono-objective lower level problem) permit solution of the above problem using classic approaches, including: direct methods [Chankong and Haimes 1983], [Mako and Haimes 1978], [Bard 1983], [Edmunds and Bard 1991]; implicit search methods [Bialas and Karwan 1984], [Candler and Townsley 1982], [Jongen and Weber 1990], [Sobol 1992]; and penalty methods [Aiyoshi and Shimizu 1984], [Installe 1994]. Additionally, classical interactive optimization procedures can be applied to solve the problem, allowing the user to investigate the various potentially satisfying alternatives [Haimes et al. 1990], [Durso 1992], [Haimes 1980], [Nijkamp and Spronk 1980], [Goicoechea et al. 1979], [Wallenius 1975].

7.3.3 Software Acquisition's Program Consequence Hierarchical Decision Problem

We focus the application of the hierarchical decision problem described in the previous section on the program consequence HHS. This decomposition constitutes the risk management decision-making problem associated with estimating and managing a software project's cost, schedule, and performance. Each of the participant communities (user, customer, contractor) contributes to resolving the competing issues of the program consequence HHS -- individually, as well as jointly.

We consider two participant community decompositions: the contractor decomposition, and the user/customer decomposition. In this formulation, the interests and objectives of the user and customer communities are combined. This is a plausible simplification, as the customer is an agent acting for, and in behalf of, the user. The customer is responsible for procuring a system that meets the user's needs, while ensuring that the procedures and requirements of the acquisition process are followed and enforced. We also consider two program consequence elements: project cost and project schedule.

The two community decompositions are interrelated through their mutual interests in the successful acceptance, continuance, and completion of the software project. More specifically, the contractor and customer communities are connected by solutions to the cost and schedule elements of the program consequence HHS that meet the specific objectives

of each community and are mutually-agreeable as to the objectives that overlap constituencies. Figure 7.8 provides a graphical depiction of the two-decomposition coordination problem.

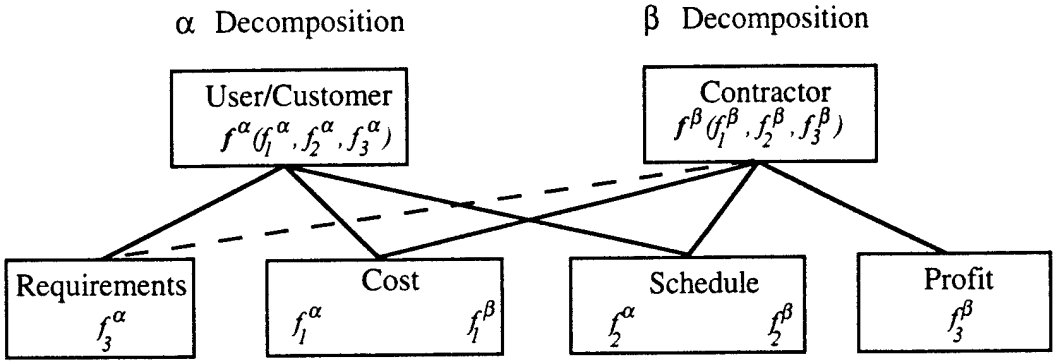


Figure 7.8 Hierarchical Decision Problem for Participant Community - Program Consequence Coordination

Each community has objectives related to the cost and schedule elements as well as its own unique objectives. The general description of the objectives, uncertain quantities, and decision variables of the hierarchical decision problem is given in Table 7.5.

Table 7.5 Description of the Hierarchical Decision Problem Formulation

α **Decomposition: User/Customer**	β **Decomposition: Contractor**
Objectives f^{α}	Objectives f^{β}
minimize: f_1^{α} - Cost overrun	minimize: f_1^{β} - Cost overrun
minimize: f_2^{α} - Schedule delay	minimize: f_2^{β} - Schedule delay
minimize: f_3^{α} - Unmet requirements	maximize: f_3^{β} - Profit
Random variable	Random variable
R- Requirement change requests	R- Requirement change requests
Decision variable x^{α}	Decision variables x^{β}
x_1 - Allowed requirements changes	x_2 - Personnel resources
	x_3 - Technology resources

7.3.4 The User/Customer Decomposition Decision Problem

The user/customer's multiobjective decision problem is focused on how to deal with the ever-increasing demand for system requirement changes. The frequency or extent of requirements changes has been shown to be a primary factor affecting software project cost

overruns and schedule delays [Lederer and Prasad 1993], [Boehm and Papaccio 1988]. Generally, increased requirements lead to a more complex, more costly system that takes longer to develop. Controlling the requirements changes has been shown to be one way in which the customer can make a significant, direct impact on managing cost overruns and schedule delays [Boehm and Papaccio 1988]. Hence, the decision variable for the customer is the allowable requirements changes. This decision is made in light of the objectives of minimizing cost overruns and schedule delays, and the desire to maximize the user's satisfaction with the system by minimizing the unmet requirements changes.

The formulation of the cost and schedule objective functions is consistent with those in previous chapters and is based on the COCOMO models. The cost overrun and schedule delay objective functions for the user/customer are the difference between the original estimates and the new estimates considering the allowed requirements changes:

$$f_1^{\alpha} = \text{cost overrun} = k_1\left[a\left(KLOC \cdot x_1\right)^b\right] - k_1\left[a\left(KLOC\right)^b\right]$$
$$= ak_1\left(KLOC\right)^b\left(x_1^b - 1\right), \tag{7.2}$$

$$f_2^{\alpha} = \text{schedule delay} = c\left[a\left(KLOC \cdot x_1\right)^b\right]^d - c\left[a\left(KLOC\right)^b\right]^d$$
$$= c\left[a\left(KLOC\right)^b\right]^d\left(x_1^{bd} - 1\right), \tag{7.3}$$

where

a,b,c,d	are system characteristic parameters as with the COCOMO
$KLOC$	is the original system size estimate
x_1	is the allowed requirements changes, quantified as a percentage KLOC multiplier ($x_1 \geq 1.0$)
k_1	is a cost-per-man-month multiplier.

Note that objective functions (7.2) and (7.3) are minimized when the decision variable x_1 equals 1.0, or when there is no increase in system size or complexity due to requirements changes that lead to cost overruns and schedule delays.

The user/customer's third objective function is to minimize unmet requirements changes. The user's desire over the life cycle for requirements changes (R) is difficult to predict and never known with certainty. Quantified as a percentage KLOC multiplier, R ≥ 1.0. With a probability distribution associated with R that captures the degree of uncertainty

concerning the user's desired requirements changes, and x_1 defined as the level of allowed requirements changes as set by the customer, an important measure of the user's eventual dissatisfaction with the completed system is the unmet requirements changes ($R - x_1$). As there is a probability distribution associated with R, then the new measure (unmet requirements) is also a probability distribution -- a translation of the desired requirements changes distribution (Figure 7.9).

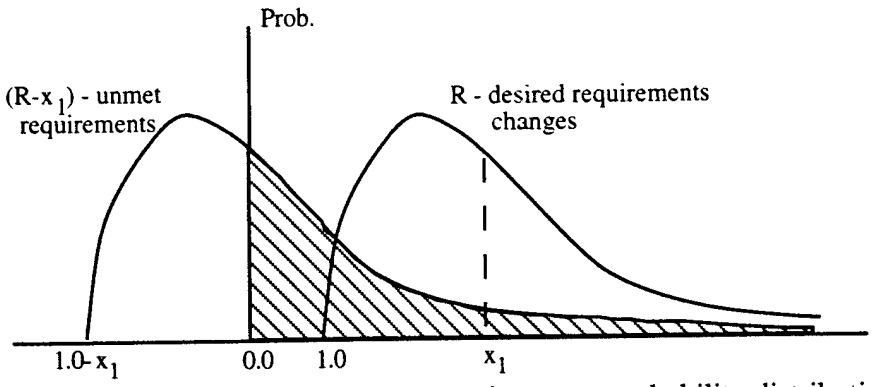


Figure 7.9 Requirements and unmet requirements probability distributions

Of most interest in the ($R - x_1$) distribution is the non-negative portion that describes the unmet requirements given the customer's decision (shaded region in Figure 7.9). There are several possible decision-making metrics related to the unmet requirements:

i) $\Pr[R - x_1 > 0]$ the probability of unmet requirements; an indication of the chance of user dissatisfaction,

ii) $E[R - x_1]$ the expected unmet requirements,

iii) $E[R - x_1 | x_1 = \beta]$ the extreme-event conditional expected unmet requirements, where β is the partitioning point associated with a particular probability partitioning value,

iv) $E[R - x_1 | R - x_1 > 0]$ the expected unmet requirements, given that there are unmet requirements.

The first measure indicates the chance of unmet requirements, while the remaining three measures are indications of the extent to which the requirements are not met. Measures (ii) and (iii) include in their calculation the portion of the distribution that is negative (requirements being fully met or exceeded) -- hence may not give an accurate representation of the magnitude of unmet requirements. Measure (iv), however, indicates

the extent of unmet requirements, given that there are some. Together, measures (i) and (iv) indicate the chance of user dissatisfaction due to unmet requirements and how great that dissatisfaction may be. These two elements are included in the two-part, third user/customer objective function:

$$f_{3a}^{\alpha} = \text{probability of unmet requirements} = \Pr\left[R - x_1 > 0\right],$$

$$f_{3b}^{\alpha} = \text{expected unmet requirements (given some exist)} = E\left[R - x_1 | R - x_1 > 0\right] \quad (7.4)$$

where

R is the desired requirements changes, quantified as a percentage KLOC multiplier

x_1 is the allowed requirements changes, quantified as a percentage KLOC multiplier ($1.0 \leq x_1$).

This third objective function can be considered a type of disutility measure, where increasing values of f_3^{α} indicate increasing dissatisfaction with the system. While Eq. (7.4) is improved by increasing x_1, doing so degrades the other two objective functions. Resolution of the user/customer's multiobjective problem will be through trade-off evaluation of the Pareto optimal solutions considering the (non-commensurate) objective functions: f_1^{α} vs f_2^{α}, f_1^{α} vs f_3^{α}, and f_2^{α} vs f_3^{α}.

7.3.5 The Contractor Decomposition Decision Problem

The contractor's multiobjective decision problem includes maximizing profit, and two objectives similar to that of the user/customer: minimize cost overrun and schedule delay. It is in the contractor's interest to consider the satisfaction of the customer, which is done by meeting the cost and schedule limitations of the original contract. Minimizing cost overruns and schedule delays also provide the contractor a good working relation with the customer, good track-record for future contracts, the possibility of incentive awards, etc. Additionally, reducing project time delays allows the contractor to re-assign personnel to other efforts, allowing the organization to take on more projects.

The contractor affects the project's cost and schedule by the personnel and technology resources employed in the development effort. Higher qualified and more capable analyst and programmer personnel may require higher personnel costs (higher salaries), however productivity is higher, which decreases the man-months of effort and development time

requirements, and may lower the overall project costs. Application of software development technologies (e.g., CASE tools, clean-room development environments, COTS, etc.) may also reduce effort and development time requirements.

The cost and schedule objective functions for the contractor consider the customer's approved requirements changes decision and the contractor's personnel and technology decisions:

$$f_1^{\beta} = \text{cost overrun} = k_2 a\left[\left(KLOC \cdot x_1\right)\left(x_2 \cdot x_3\right)\right]^b - k_1 a\left(KLOC\right)^b$$
$$= a\left(KLOC\right)^b\left[k_2\left(x_1 x_2 x_3\right)^b - k_1\right], \tag{7.5}$$

$$f_2^{\beta} = \text{schedule delay} = c\left[a\left[\left(KLOC \cdot x_1\right)\left(x_2 \cdot x_3\right)\right]^b\right]^d - c\left[a\left(KLOC\right)^b\right]^d$$
$$= c\left[a\left(KLOC\right)^b\right]^d\left[\left(x_1 x_2 x_3\right)^{bd} - 1\right], \tag{7.6}$$

where

a,b,c,d	are system characteristic parameters as with the COCOMO
$KLOC$	is the original system size estimate
x_1	is the customer-determined allowed requirements changes, quantified as a percentage KLOC multiplier ($x_1 \geq 1.0$)
x_2	denotes personnel experience and capability, quantified as a percentage KLOC multiplier ($0.5 \leq x_2 \leq 1.5$, nominal = 1.0)
x_3	denotes the software development technology resources, quantified as a percentage KLOC multiplier ($0.5 \leq x_3 \leq 1.5$, nominal = 1.0)
k_1	is the original cost-per-man-month multiplier
k_2	is the new cost-per-man-month multiplier considering the new personnel and technology resource decisions.

The contractor's profit function, formulated according to a cost-plus development contract, is given by

$$f_3^{\beta} = \text{profit} = p\left(k_2 a\left[\left(KLOC \cdot x_1\right)\left(x_2 \cdot x_3\right)\right]^b\right), \tag{7.7}$$

where all terms are defined as above, and

p	is the percentage of costs allowed for profit (cost-plus percentage).

Note that while Eqs. (7.5) and (7.7) are minimized when the product of the decision variable values equals 1.0, Eq. (7.7) provides no such incentive for improving personnel or technology resources to compensate for the customer's requirement changes. In fact, with a cost-plus objective function the contractor desires that the customer make more requirements changes -- doing so increases the overall costs and the contractor's profit.

7.3.6 The Program Consequences Decomposition Hierarchical Decision Problem

The program consequences decomposition hierarchical decision problem, in the form of Section 7.3.2 where the user/customer constitutes one decision making decomposition and the contractor the other, is given by:

Find $\boldsymbol{x}^{\alpha}$ the best compromise with respect to $\boldsymbol{f}^{\alpha}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha}))$
such that $\boldsymbol{x}^{\alpha} \in \mathrm{X}^{\alpha}$ and $\mathbf{g}^{\alpha}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})) \leq 0$

where $\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$ is the solution of the following problem:

Given $\boldsymbol{x}^{\alpha}$, find $\boldsymbol{x}^{\beta}$ the best compromise with respect to $\boldsymbol{f}^{\beta}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta})$
such that $\boldsymbol{x}^{\beta} \in \mathrm{X}^{\beta}$ and $\mathbf{g}^{\beta}(\boldsymbol{x}^{\alpha}, \boldsymbol{x}^{\beta}) \leq 0$ (7.8)

where

$\boldsymbol{x}^{\alpha}$	consists of x_1, the customer decision variable
$\boldsymbol{x}^{\beta}$	consists of x_2, x_3, the contractor decision variables
$\boldsymbol{x}^{\beta*}(\boldsymbol{x}^{\alpha})$	stands for the reaction of the contractor, given $\boldsymbol{x}^{\alpha}$
$\boldsymbol{f}^{\alpha}$	is the customer objectives, $f_1^{\alpha}, f_2^{\alpha}, f_3^{\alpha}$
$\mathbf{g}^{\alpha}$	is the customer constraints
X^{α}	is the customer definition set, $\boldsymbol{x}^{\alpha} \geq 1.0$
$\boldsymbol{f}^{\beta}$	is the contractor objectives, $f_1^{\beta}, f_2^{\beta}, f_3^{\beta}$
$\mathbf{g}^{\beta}$	is the contractor constraints
X^{β}	is the contractor definition set, $0.5 \leq \boldsymbol{x}^{\beta} \leq 1.5$.

7.3.7 Solution Procedure for the Hierarchical Bi-element Decision Problem

To solve the bi-element problem, we must be able to compute individually the solution of both decision problems [Installe 1994]. In other words, the solution approach for the two-decomposition problem of the user/customer and contractor requires the solution of the

independent problems and then a coordinated resolution of their results. We first outline the solution approach for the individual multiobjective decision problems and then introduce approaches for a coordinated solution.

7.3.7.1 Solution Procedure for the User/Customer Decomposition. The customer determines allowed requirements changes based on Pareto optimal solution trade-offs among objective functions $f_1^{\alpha}, f_2^{\alpha}, f_3^{\alpha}$. In particular, increasing levels of requirements changes lead to greater user satisfaction with the system, but simultaneously leads to greater cost overruns and schedule delays. Subjective trade-offs between the user's dissatisfaction (unmet requirements changes) and cost overruns, and between unmet requirements changes and schedule delays leads to the selection of a preferred solution from the set of Pareto optimal solutions (Figure 7.10). Selection of the desired option then fixes the value of the requirements changes decision variable.

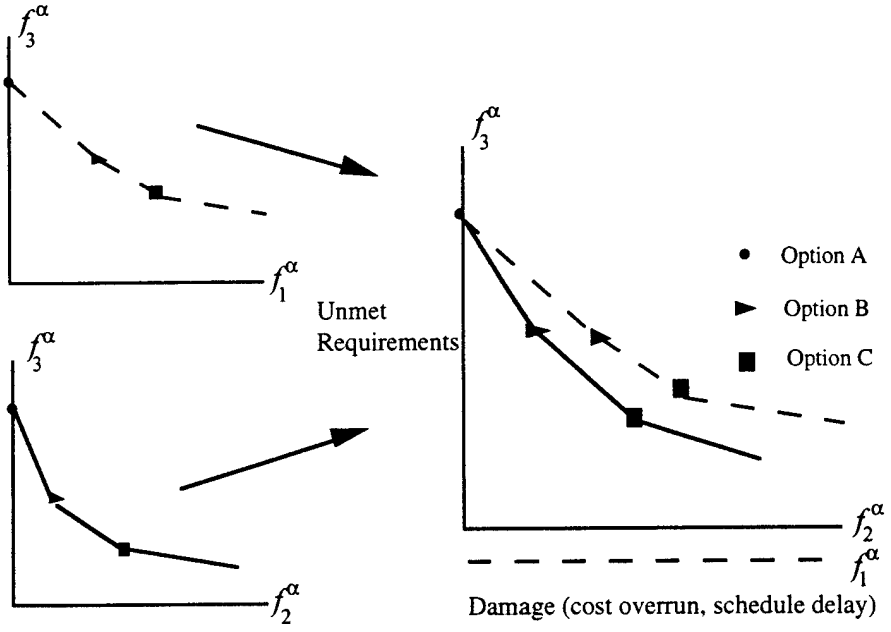


Figure 7.10 User/Customer Pareto optimal trade-offs

7.3.7.2 Solution Procedure for the Contractor Decomposition. The individual solution to the contractor's problem is similar to that of the customer's. Given the customer's requirements change level, the contractor determines the personnel and technology resource allocations based on Pareto optimal solution trade-offs among objective functions $f_1^{\beta}, f_2^{\beta}, f_3^{\beta}$. In particular, increasing the technology quality or personnel experience leads to a reduction in development schedule, but at a greater cost.

The increased cost, however, also increases the contractor's profit when assuming a cost-plus arrangement. Decreasing the quality of technology and personnel resources may reduce per-month resource costs, but these are often offset by the lengthened schedule. Subjective trade-offs between the customer's satisfaction (minimizing cost overrun and schedule delay) and maximizing contractor profit (or the equivalent, minimizing negative profit) lead to the selection of a preferred solution from the set of Pareto optimal solutions (Figure 7.11). Selection of the desired option then fixes the value of the personnel and technology resource allocation decision variable.

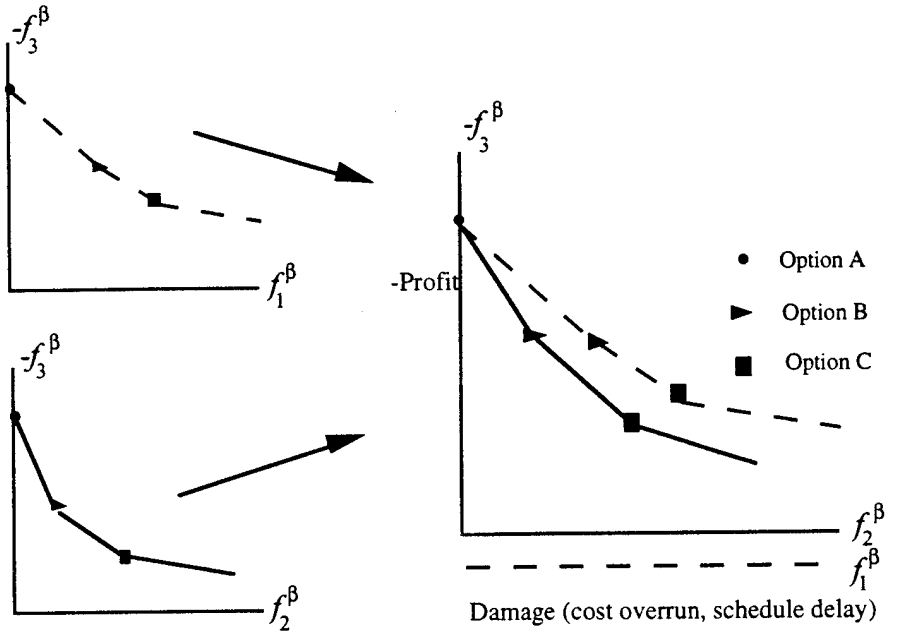


Figure 7.11 Contractor Pareto optimal trade-offs

7.3.7.3 Example 7.2 - The Hierarchical Decision Problem Approach Applied to the User/Customer and Contractor Decompositions. Consider again the software development problem of Example 7.1. Assuming an initial $3000 per man-month cost, this 50-KLOC, embedded mode project has an original project cost estimate of $918,000 and a development time of 15.6 months. The user organization' s desired requirements changes (a factor increase over the originally-projected system size), R, is quantified as a triangular distribution with a low value of 1.0, high value of 5.0 , and most likely value of 4.0. The customer must trade-off meeting all of the user's requests, while keeping project costs and schedule in control. The range of the customer's decision variable is $1.0 \leq x_1 \leq 5.0$, where the lower limit indicates no change in requirements (not allowing any of the change requests) and the upper limit indicates full compliance with the

entire possible range of the user's request. Table 7.6 summarizes the values of the three objective functions for varying levels of allowed requirements changes.

Table 7.6 Customer objective function values, varying x_1

Allowed Requirements (KLOC multiplier) x_1	Cost overrun (dollars) f_1^α Eq. (7.2)	Schedule delay (months) f_2^α Eq. (7.3)	Unmet Requirements Prob. f_{3a}^α $\Pr[R\text{-}x_1>0]$	Unmet Requirements Expect. f_{3b}^α $E[R\text{-}x_1 \mid R\text{-}x_1>0]$
1.00	0.00	0.00	1.000	2.333
1.50	575,583.44	2.63	0.979	1.876
2.00	1,191,560.46	4.76	0.917	1.485
2.50	1,839,430.05	6.58	0.813	1.141
3.00	2,513,903.97	8.19	0.667	0.833
3.50	3,211,337.10	9.64	0.479	0.558
4.00	3,929,047.54	10.97	0.250	0.333
4.50	4,664,970.27	12.20	0.125	0.167

Observe that complying with nearly all of the user's requirements change requests would lead to a cost overrun more than five times the project's original cost and would have a schedule delay eighty percent again as long as the original project schedule. A plot of the expected unmet requirements changes versus the other two objective functions allows graphical consideration of the customer's Pareto optimal alternatives (Figure 7.12). From this plot one observes the relationship between the unmet requirement changes and the other two objective functions. Interactive trade-off approaches such as the surrogate worth trade-off (SWT) [Haimes 1981] can be applied to select the most desirable solution. Selection of a desired solution then fixes the value of the decision variable x_1.

In response to the Customer's determination of x_1 (allowed requirement changes), the contractor must trade-off meeting the cost and schedule requirements while maximizing profit. The contractor's personnel and technology decision variables have a schedule compression effect as well as a cost effect -- more experienced personnel are more productive and the project can be accomplished in less time. This a schedule savings comes at a cost, however, as more experienced personnel mean higher personnel costs. A

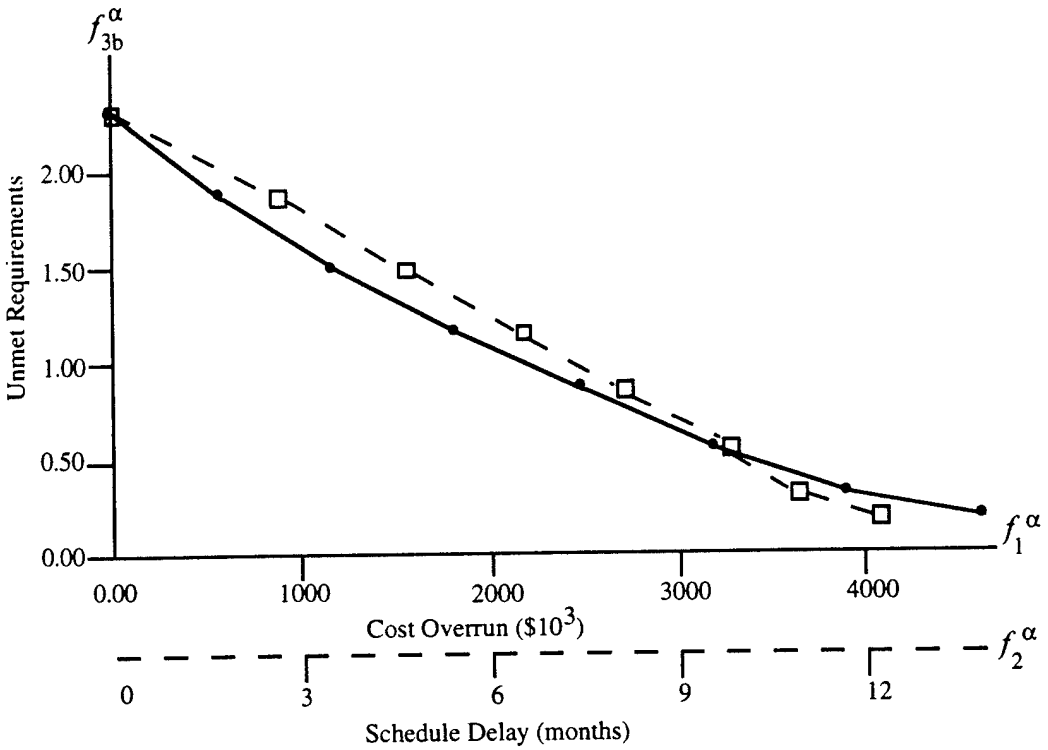


Figure 7.12 Customer's Pareto optimal solutions

parallel argument can be made for advanced technologies - using better technologies increases cost, but can reduce development time. On the other hand, less expensive but less capable personnel (technologies) lead to a longer development time and the anticipated cost savings may be negated by the longer schedule.

The range of the contractor's decision variables is set at $0.5 \leq x_2, x_3 \leq 1.5$, implying the limitations that personnel and technology can have on affecting project development effort and schedule. A decision variable value of 1.0 implies the nominal resource level. Higher decision variable multiplier values indicate degraded technology or less-experienced personnel that lead to higher effort requirements; lower decision variable multiplier values indicate a reduction in effort requirement due to better technology or experienced personnel.

Table 7.7 summarizes the values for the contractor's three objective functions for varying levels of allowed requirements changes and personnel and technology resources. The resource cost multiplier k_2, is associated the personnel and technology decision. The cost multiplier for less-experienced personnel decreases less rapidly than does the increasing

Table 7.7 Contractor objective function values, varying x_1, x_2, x_3

Allowed Req'mts x_1	Personnel/ Technology $x_2 \bullet x_3$	Resource cost multiplier k_2	Cost overrun (dollars) f_1^β Eq. (7.5)	Schedule delay (months) f_2^β Eq. (7.6)	Profit (dollars) f_3^β Eq. (7.7)
1.00					
	0.50	7,500	80,994.18	-3.65	99,941.83
	0.75	4,500	57,032.72	-1.63	97,545.69
	0.90	3,500	25,813.03	-0.62	94,423.72
	1.00	3,000	0.00	0.00	91,842.41
	1.10	2,750	25,475.79	0.58	94,389.99
	1.25	2,500	81,930.66	1.40	100,035.48
	1.50	2,250	202,081.55	2.63	112,050.57
1.50					
	0.50	7,500	707,337.29	-1.63	162,576.14
	0.75	4,500	668,359.00	0.72	158,678.31
	0.90	3,500	617,573.69	1.91	153,599.78
	1.00	3,000	575,583.44	2.63	149,400.76
	1.10	2,750	617,025.10	3.31	153,544.92
	1.25	2,500	708,860.67	4.26	162,728.48
	1.50	2,250	904,311.04	5.70	182,273.52
2.00					
	0.50	7,500	1,377,636.21	0.00	229,606.04
	0.75	4,500	1,322,587.23	2.63	224,101.14
	0.90	3,500	1,250,863.22	3.95	216,928.74
	1.00	3,000	1,191,560.46	4.76	210,998.46
	1.10	2,750	1,250,088.45	5.52	216,851.26
	1.25	2,500	1,379,787.69	6.58	229,821.18
	1.50	2,250	1,655,821.94	8.19	257,424.61
2.50					
	0.50	7,500	2,082,640.26	1.40	300,106.44
	0.75	4,500	2,010,688.52	4.26	292,911.27
	0.90	3,500	1,916,941.70	5.70	283,536.58
	1.00	3,000	1,839,430.05	6.58	275,785.42
	1.10	2,750	1,915,929.02	7.41	283,435.32
	1.25	2,500	2,085,452.34	8.57	300,387.65
	1.50	2,250	2,446,242.76	10.32	336,466.69

cost of advanced technology and experienced personnel. As anticipated, improved resources result in schedule compression (hence, less schedule delay) and increased cost. It can also be noted that an attempt at cost reduction through less expensive personnel and technology is, instead, offset by the longer development schedule.

Selecting the scenario for which the customer's decision variable is set at a particular level, we can analyze the contractor's associated decision problem. A plot of the negative profit objective function values (negative, so all objectives are now to be minimized) versus the cost overrun and schedule delay objective function values for $x_1 = 1.50$ shows the Pareto optimal alternatives for the contractor (Figure 7.13).

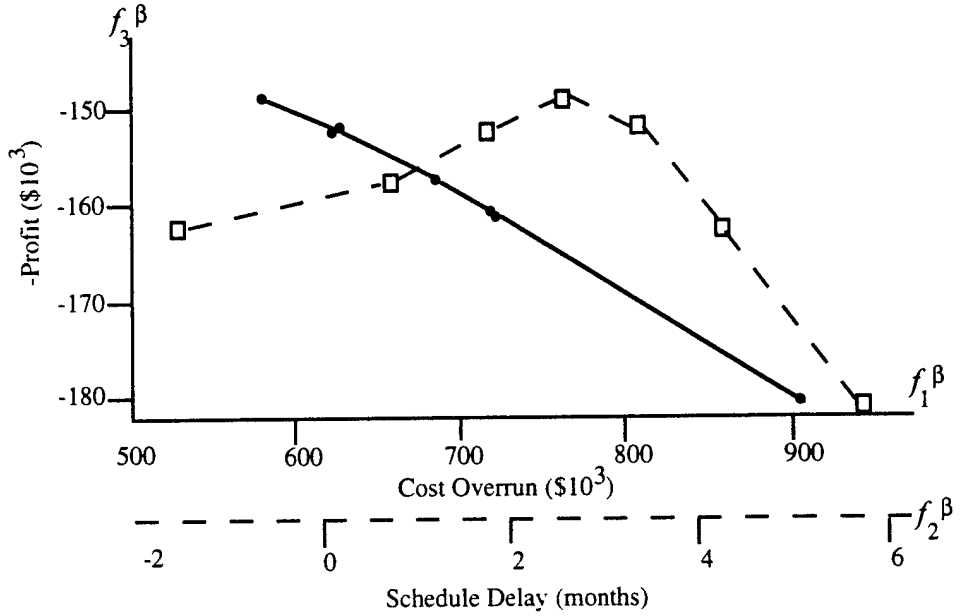


Figure 7.13 Contractor's Pareto optimal solutions ($x_1 = 1.50$)
(note: vertical axis indicates the minimization of negative profit)

With the cost-plus contract, the contractor's profit increases as cost increases -- hence the addition of more costly personnel and technology increases profit; interestingly enough, the longer development time required by less-experienced personnel has a similar effect on profit. In light of the other objectives, however, those alternatives that include improved personnel and technology would be preferred as these also lead to reduced schedule delays.

7.3.8 Negotiation and Convergence for the Hierarchical Decision Problem

Resolution of the two decomposition solutions is possible through negotiated iteration, with consideration of the (possibly subjective) trade-off information for each decision maker. After each decision maker optimizes his or her subproblem, the trade-offs between objective functions for each decision maker can be calculated as in Section 6.3.2 (e.g., trade-off between f_1^α and f_1^β). Then, based on this trade-off information, the decision makers agree on the direction and amount of change in f_1^α and f_1^β (using his or her own

decision rule). Now, after observing the contractor's response, the customer adjusts the original decision in light of this new information. Then, with the customer's revised solution, the contractor is also able to adjust the former solution.

The trade-off between first objectives (cost overrun), selected for its consistent formulation between the subproblems, is a simple measure of the interdependence of the decision makers. Note that each decision maker's reaction to that trade-off may also depend naturally on the levels of all objectives. With the other community's trade-off information, negotiations can lead to a mutually-agreeable solution.

Concerning convergence of the process, "an iterative negotiation scheme converges if the objective functions converge to some limits when the process is continued by consistent decision makers, and if the limit values of the objectives are a best compromise among all feasible objective values for the decision makers" [Haimes et al. 1990]. To fully consider the convergence of the iterative scheme, one must address several assumptions [Sage 1977]: the existence of a best-compromise solution, the existence of a utility function, existence of all trade-offs between objectives, etc. Convergence may also be considered in less-rigid terms, such as the continued reduction in the difference between the subproblem solutions, percentage improvement in objective function values, etc.

7.4 Chapter Summary

In this Chapter we have made two important contributions to the software estimation methodologies developed in previous chapters: i) an approach for updating software estimates throughout the life cycle in light of actual project progress , and ii) formulation and resolution of a hierarchical decision making problem that manages the competing issues and objectives within and among participant communities.

The HHM provides an ideal framework for evaluating the cause for deviation between actual and estimated project development effort and schedule requirements. Accounting for the cause for the discrepancy permits appropriate adjustments to be made to the estimation models. The HHM allows for a wide range in the level of detail considered in such on-going analysis. Systematic updating of the software estimates provides an improved indication of a software project's progress relative to its estimated completion, along with a way for analyzing those factors which contribute to deviation from previous estimates.

Finally, the bi-element hierarchical decision-making formulation provides increased insight and understanding, along with improved decision-making correlation among the various decompositions of the software acquisition HHM framework. While this Chapter applied the hierarchical decision problem approach to the user/customer and contractor decompositions, the decision problems associated with other decompositions can be represented through similar formulation. Each decomposition's multiobjective decision making problem includes distinct, as well as overlapping objectives and other model elements. The decisions of each decomposition affect the others, and no decomposition has control of all decisions and other model elements. The problem formulation provided in this Chapter permits independent and coordinated solutions, while increasing the understanding of the interactions between the decompositions. By applying this method, mutually-agreeable solutions can be obtained.

Chapter 8
Summary, Conclusions, and Future Work

8.1 Summary and Conclusions of the Dissertation

This dissertation addresses the assessment and management of risks associated with the software acquisition processes from a holistic perspective using hierarchical holographic modeling (HHM). The multiple visions and perspectives within which the life cycle of software acquisition is stated and modeled, provide a comprehensive framework for risk assessment and management of software acquisition. In particular, widely used models in software acquisition such as the COCOMO model, can now be extended to incorporate probabilistic as well as dynamic dimensions. The ultimate contributions of this dissertation can be found in at least two major areas: (a) in the theoretical and methodological domain of systems modeling in the quest of a more quantitative risk assessment and management framework, and (b) in advancing the state of practice in the assessment and management of software acquisition by extending highly used models in practice to incorporate more realistic probabilities and dynamic dimensions.

A holistic framework for risk assessment and management that provides a comprehensive structure for identifying risk sources, assessing and measuring the risks, explicitly considering inherent uncertainties, and resolving competing objectives and issues among participant communities and other decompositions, is important to improving software acquisition management. The potential exists for a positive improvement in software acquisition management by maturing the capabilities of the customer community through the development of theoretical and methodological foundations. New software estimation and software acquisition decision-making methodologies developed in this research address the customer's capability to identify and assess the programmatic risks associated with software acquisition and to make more-informed control policy and resource allocation decisions.

As the customer community is, in general, in the process of maturing from very low levels of software acquisition knowledge, analytical methods for software acquisition management must be appropriate for the customer's capabilities and needs. The

contributions of this research are for use at several levels of software acquisition maturity, and build on one another in terms of complexity and detail.

The general HHM framework for software acquisition was developed in Chapter 3. This model provides the framework for a comprehensive investigation of risk sources for software acquisition and leads to the development and interpretation of software acquisition analytic models, including software cost and schedule estimation models.

In Chapter 4, the theoretical contribution of the triangular distribution for extreme event analysis was enhanced by the derivation and analysis of closed-form solutions for the conditional expectation functions of the triangular distribution. These results were then used in developing a probabilistic software estimation approach in Chapter 5 -- both a direct probabilistic estimation approach, and an approach that uses Monte Carlo simulation in producing the estimate. The probabilistic approach advances current practices in software estimation by explicitly considering the uncertainty associated with estimating a software project's cost and schedule, and by utilizing the conditional expected value as a supplementary decision-making metric. Also in Chapter 5, a method for calculating the conditional and unconditional expected values when using Monte Carlo simulation was developed as part of the probabilistic software estimation methodology. Deploying the approach to the COCOMO model demonstrated the benefits of the probabilistic software estimation approach.

In Chapter 6, dynamic software estimation models were developed that advanced the state-of-the-art in software estimation to meet the analytic requirements of the most-current spiral and prototyping software development process paradigms. These models account for the dynamics of changing requirements, system design, and other policy factors. The dynamic formulation permits analysis of the effect of current-stage decisions on future decision opportunities in light of the multiple objectives associated with cost overrun and schedule delay.

Chapter 7 addressed two issues associated with the dynamic models: updating software estimates in light of actual project progress and resource expenditures; and coordinating and resolving competing issues, objectives, and decision opportunities among the participant communities. Development of a software estimation updating scheme provided the framework for on-going improvement in estimating a project's cost and schedule requirements. Then, a hierarchical decision problem formulation was devised that allowed

for equal-level interaction among hierarchical sub-problems to resolve the overlapping and unique issues of the sub-problems. Deployment to the software acquisition user, customer, and contractor decision problems demonstrated the approach for resolving the individual multiobjective sub-problems and for coordination among the sub-problems that leads to a mutually-agreeable solution.

The multiobjective problems of the user/customer and contractor decompositions provide greater insight concerning the interactions and effects of each community's decisions, particularly concerning the effect of requirements changes and resource allocation policies on project cost overrun and schedule delay.

8.2 Recommendation for Future Work

This section outlines four areas for future work that would extend the theoretical and methodological contributions of this research.

8.2.1 Determine the Functional Form of the Time-Variant Coefficients in the Dynamical Software Estimation Model.

The deployment of the dynamic model in Chapter 6 assumed constant values over time for several of the model parameters. In practice, this assumption may be unrealistic. For example, programmer capabilities improve over the development period as the programmers become more familiar with the project, the language used, etc. Accounting for such 'learning curves' and other dynamic tendencies would improve the software estimate. The functional forms of some model elements could be derived through examining existing datasets of completed software projects.

8.2.2 Evaluate the Effect of Up-Front Expenditures on Overall Project Costs Using the Dynamic Model

Krishnan and Kellner [1995] [1993] have explored the empirical relationship between software cost and product quality, and between software life-cycle costs and front-end expenditures. Further exploration of the impact of early-life cycle requirements, design, and expenditure policies on life-cycle costs is possible by using the dynamic software estimation model. Additionally, the dynamic software estimation model could be used to

explore the effects of personnel characteristics, tool deployment, and other factors on life-cycle costs.

8.2.3 Include Bayesian Revision in the Dynamical Software Cost Estimation Model

The dynamical software estimation model might possibly be reformulated as a sequential forecast process ([Katz et al. 1992], [Krzysztofowicz and Davis 1983]), not unlike the flood forecasting problem [Li et al. 1992]. In a pure sequential forecast process, forecasts of a fixed but uncertain state are prepared with decreasing lead times, with each subsequent forecast incorporating additional information and, therefore, updating the previous forecast. The real thousands of lines of code (KLOC) in the software estimation problem acts as the real flood peak in the flood forecasting problem, whose true value is hidden and can only be known with certainty after the whole process is realized. However, at every time period before the final stage, there is an attainable *estimation* of the hidden variable. Each stage's estimate of KLOC could be used to update the probabilistic description of the true KLOC through Bayesian revision. It may then be possible to derive a probabilistic description of project development effort from the posterior density function and use the conditional expected value of project cost as a supplement to the traditional expected value for decision making.

8.2.4 Analyze Contract Vehicle Options and Requirements Change Policies Using the Hierarchical Decision Problem Formulation

The hierarchical decision problem formulated in Chapter 7 assumed a cost-plus contract arrangement for the contractor, where profit is a percentage of development costs. The hierarchical decision-making approach can be used to analyze other contracting options (e.g., incentive contracting, fixed-price contracting) as they impact project cost overrun and schedule delay.

The user/customer - contractor hierarchical decision problem formulated in Chapter 7 identified the significance of requirements changes as a principal factor contributing to project cost overrun and schedule delay. The decision problem can be used to provide additional investigation regarding requirement change policies, their impact at different points in the life cycle, etc. on cost overrun and schedule delay.

References

Abbott, R.J. *An Integrated Approach to Software Development*, Wiley, New York, 1986.

Abdel-Hamid, T.K. *The Dynamics of Software Development Project Management: An Integrative Systems Perspective*, Ph.D. thesis, Sloan School of Management, MIT, Cambridge, MA, 1984.

Agresti, W.W., and W.M. Evanco. "Projecting Software Defects from Analyzing Ada Designs," *IEEE Transactions on Software Engineering*, 18, 988-997, 1992.

Aiyoshi, E., and K. Shimizu. "A solution method for the static constrained problem via penalty methods," *IEEE Transactions on Automatic Control*, 29, 1111-1114, 1984.

Albrecht, A.J., and J.E. Gaffney. "Software Function, Source Lines Of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering* SE-9, 639-648, 1983.

Applegate, L.M., B.R. Konsynski, and J.F. Nunamaker. "Model Management Systems Design for Decision Support," *Decision Support Systems*, 2(1), 81-91, 1986.

Asbeck, E., and Y. Y. Haimes. "The partitioned multiobjective risk method," *Large Scale Systems* 6(1), 13-38, 1984.

Augustine, Norman R. *Augustine's Laws*, American Institute of Aeronautics and Astronomics, New York, 1993.

Austin, R.D., and D.J. Paulish. *A Survey of Commonly Applied Methods for Software Process Improvement,* Technical Report CMU/SEI-93-TR-27, Software Engineering Institute, Carnegie Mellon University, Pitsburgh, PA, 1993.

Avizienis A. "The N-Version Approach to Fault-Tolerant Software," *IEEE Transactions on Software Engineering*, SE-11(12), 1491-1501, Dec. 1985.

Bailey, E.K. *A Descriptive Evaluation of Automated Software Cost Estimation Models,* Institute for Defense Analysis, Alexandria Va, Oct. 1986.

Baker, E.R., L. Cooper, B.A. Corson, A.E. Stevens. "Software Acquisition Management Maturity Model (SAM3)," *Program Manager*, 43-49, July-August 1994.

Banker, R., and K. Kemerer. "Scale Economics in New Software Development," *IEEE Transactions on Software Engineering*, 15(10, 1199-1205, 1989.

Bard, J.F. "An efficient point algorithm for a linear two-stage optimization problem," *Operations Research*, 31, 670-684, 1983.

Bard, J.F., and J.T. Moore. "A branch and bound algorithm for the bilevel programming problem," *S.I.A.M. Journal of Scientific and Statistical Computing*, 11, 281-292, 1990.

Barros, O. "A Pragmatic Approach to Computer-Assisted Systems Building," *Journal of Systems and Software* 18, 1992.

Barrow, D., S. Nilson, and D. Timberlake. *Software Estimation Technology Report*, Software Technology Support Center (STSC), Hill AFB, UT, 1993.

Basu, A., and R.W. Blanning. "Model Integration Using Metagraphs," *Information Systems Research*, 5(3), 195-218, Sept. 1994.

Bell, G.A. "Applying the System Design Dynamics Technique to the Software Cost Problem: A Rationale," *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Benbasat, I., and I. Vessey. "Programmer and Analyst Time/Cost Estimation," *MIS Quarterly*, 4(2), 30-43, 1980.

Bertsekas, D.P. *Dynamic Programming and Stochastic Control*, Academic Press, New York, 1976.

Bialas, W.F., and M.H. Karwan. "On two-level optimization," *IEEE Transactions on Automatic Control*, 27, 211-214, 1982.

Bialas, W.F., and M.H. Karwan. "Two-level linear programming," *Management Science*, 30, 1004-1020, 1984.

Bittanti, S., P. Bolzern, and R. Scattolini. "An Introduction to Software Reliability Modeling," in *Software Reliability Modeling and Identification*, G. Goos and J. Hartmanis (eds.), Springer-Verlag, New York, 1988.

Blackwell, D., and M.A. Girshick. *Theory of Games and Statistical Decisions*, Wiley, New York, 1954.

Blanning, R.W. "A Relational Framework for Joint Implementation in Model Management Systems," *Decision Support Systems,* 1(1), 69-81, Jan. 1985.

Blum, B. I. *Software Engineering, A Holistic View*, Oxford University Press, New York, 1992.

Boehm, B.W. *Software Engineering Economics*, Prentice Hall, Englewood Cliffs, NJ, 1981.

Boehm, B.W. "Verifying and Validating Software Requirements and Design Specification," *Software*, 75-88, Jan. 1984.

Boehm, B.W. "A Spiral Model of Software Development and Enhancement," *Computer* 21(5), 61-72, 1988.

Boehm, B.W. *Software Risk Management*, IEEE Computer Society Press, Washington D.C., 1989.

Boehm, B.W., B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Shelby. "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," to appear in *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, J.D. Arthur and S.M. Henry (eds.), J.C. Baltzer AG Science Publishers, Amsterdam, 1995.

Boehm, B.W., and P.N. Papaccio. "Understanding and Controlling Software Costs," *IEEE Transactions on Software Engineering*, 14(10), 1462-1477, 1988.

Boehm, B.W., and W. Royce. "Ada COCOMO: TRW IOC Version," *Proceedings of the Third COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1987.

Bonczek, R.H., C.W. Holsapple, and A.B. Whinston. "The Evolving Roles of Models in Decision Support Systems," *Decision Sciences*, 11, 337-356, Apr. 1981.

Brown, A.W., D.J. Carney, E.J. Morris, D.B. Smith, and P.F. Zarrella. *Principles of CASE Tool Integration*, Oxford University Press, New York, 1994.

Candler, W., and R. Townsley. "A linear two-level programming problem," *Computer and Operations Research*, 9, 59-76, 1982.

Carr, M.J. "A Circular Model for the Complete Software Life-Cycle," *AIAA Computers in Aerospace VII Conference*, Monterey, CA, Oct. 1989.

Carr, M.J., S.L. Konda, I. Monarch, F.C. Ulrich, and C.F. Walker. *Taxonomy-Based Risk Identification*, Technical Report CMU/SEI-93-TR-06, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, June 1993.

Chankong, V., and Y.Y. Haimes. *Multiobjective Decision Making*, North-Holland, New York, 1983.

Chen, C.I., and J.B. Cruz. "Stackelberg solution for two person games with biases information," *IEEE Transactions on Automatic Control*, 17, 791-798, 1972.

Chittister, C., and Y.Y. Haimes. "Risk Associated with Software Development: A Holistic Framework for Assessment and Management," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-23(3), May/June 1993.

Chittister, C., and Y.Y. Haimes. "Assessment and Management of Software Technical Risk," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-24(2), Feb. 1994.

Chittister, C., and Y.Y. Haimes. "Systems Integration via Software Risk Management," to appear in *IEEE Transactions on Systems, Man, and Cybernetics*, 1995/1996.

Conte, S.D., H.E. Dunsmore, and V.Y. Shen. *Software Engineering Metrics and Models*, Benjamin/Cummings, Menlo Park, CA, 1986.

Davis, A.M., E.H. Bergoff, and E.R. Comer. "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Transactions on Software Engineering*, 14, , 1453-1461, Oct. 1988.

Defense Science Board (DSB). *Report of the Defense Science Board Task Force on Military Software*, Office of the Under Secretary of Defense for Acquisition, Department of Defense, Washington D.C., 6-7, Sept. 1987.

Defense Science Board (DSB). *Report of the Defense Science Board Task Force on Acquiring Defense Software Commercially*, Office of the Under Secretary of Defense for Acquisition and Technology, Washington D.C., June 1994.

Department of Defense (DoD). *Defense System Software Development*, MIL-STD-2167, Office of the Under Secretary of Defense for Acquisition, Department of Defense, Washington D.C., 1988.

Department of Defense (DoD). *Defense Acquisition Management Policies and Procedures*, DODI-5000.2, Office of the Under Secretary of Defense for Acquisition, Department of Defense, Washington D.C., Feb. 1991.

Department of Defense (DoD). *Directive 5000.1, Defense Acquisition*, Office of the Undersecretary of Defense (Acquisition), U.S. Department of Defense, Washington D.C., 1991.

Department of Defense (DoD). *Software Development and Documentation*, MIL-STD-498, Department of Defense, Washington D.C., 1994.

Doerflinger, C.W., and V.R. Basili. "Monitoring software development through dynamic variables," *IEEE Transactions on Software Engineering*, SE-11(9), 978-985, Sept. 1985.

Durso, A. "An interactive combined branch-and-bound/Tchebycheff algorithm for multiple criteria optimization," *Multiple Criteria Decision Making: Theory and applications in Business, Industry, and Government, Proceedings of the Ninth International Conference*, (A. Goicoechea, L. Duckestein, and S. Zionts, eds.), Springer-Verlag, New York, 107-122, 1992.

Dutta, A., and A. Basu. "An Artificial Intelligence Approach to Model Management in Decision Support Systems," *IEEE Computer,* 17(9), 89-97, Sept, 1984.

Dutta, A., and S. Mitra. "Integrating Heuristic Knowledge and Optimization Models for Communication-Network Design," *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 999-1017, Dec. 1993.

Edmunds, T.A., and J.F. Bard. "Algorithms for non-linear bilevel mathematical programs," *IEEE Transactions on Systems, Man, and Cybernetics*, 21, 83-89, 1991.

Elam, J.J., J.C. Henderson, and L.W. White. "Model Management Systems: An Approach to Decision Support in Complex Organizations," *Proceedings of the First International Conference on Information Systems*, 98-110, Dec. 1980.

Emrick, R.D. "In search of a better metric for measuring productivity of application development," *International Function Point Users Group Conference Proceedings*, 1987.

Evanco, W.M., and R. Lacovara. "A Model-Based Framework for the Integration of Software Metrics," *Journal of Systems Software*, 26, 77-86, 1994.

Fairley, R. "Risk Management for Software Projects," *IEEE Software*, 57-67, May 1994.

Fedorowicz, J., and G.B. Williams. "Representing Modeling Knowledge in an Intelligent Decision Support System," *Decision Support Systems*, 2(1), 3-14, Jan. 1986.

Feiler, P.H. and W. Humphrey. *Software Process Development and Enactment: Concepts and Definitions*, Technical Report CMU/SEI-92-TR-4, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.

Ferens, D.V. "Software Support Cost Models: Quo Vadis?", *Journal of Parametrics*, 4(4), 64-99, Dec. 1984, .

Ferguson, J., J. Cooper, M. Falat, M. Fisher, A. Guido, J. Marciniak, J. Matejceck, and R. Webster. *Software Acquisition Maturity Model (SAMM) Version 00.01*, Draft Report SAMM-94-02, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, June 1995.

Fisher, R. *Getting to Yes: Negotiating Agreement Without Giving In*, Houghton Mifflin, Boston, 1981.

Florac, W.A. *Software Quality Measurement: A Framework for Counting Problems and Defects*, Technical Report CMU/SEI-92-TR-22, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.

Freiman, F.R., and R.E. Park. "Price Software Model - Version 3: An Overview," Proceedings, *IEEE - Workshop on Quantitative Software Models*, 32-41, Oct. 1979.

Galorath Associates. *SEER User's Manual*, Galorath Associates, Los Angeles, CA, 1989.

Gehani, N. and A. McGettrick. *Software Specification Techniques*, Addison-Wesley, 1986.

General Accounting Office (GAO). *DoD Embedded Computers*, GAO/IMTEC-90, Washington D.C., April 1990.

General Accounting Office (GAO). *Embedded Computer Systems: Significant Software Problems on C-17 Must Be Addressed*, GAO/IMTEC-92-48, Washington, D.C., May 1992a.

General Accounting Office (GAO). *Weapons Acquisition, A Rare Opportunity for Lasting Change*, GAO/NSIAD-93-15, Government Printing Office, Washington D.C., December 1992b.

Ghezzi, C., A. Morzenti, and M. Pezze. "On the Role of Software Reliability in Software Engineering," in *Software Reliability Modeling and Identification*, G. Goos and J. Hartmanis (eds.), Springer-Verlag, New York, 1988.

Glass, R.L. *Software Reliability Guidebook*, Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Goicoechea, A., L. Duckstein, and M.M. Fagel. "Multiple objectives under uncertainty: An illustrative application of PROTRADE," *Water Resources Research*, 15, 203-210, 1979.

Gomide, F., and Y.Y. Haimes. "The multiobjective multistage impact analysis method: theoretical basis," *IEEE Transactions on Systems, Man, and Cybernetics* SMC-14(1), 88-98, 1984.

Haimes, Y.Y. "The surrogate worth trade-off method and its extensions," in *Multiple Criteria Decision Making Theory and Applications -- Hagen/Konigswinter*, West Germany 1979 (G. Fandel and T. Gal, eds.), Springer, Berlin, 85-108, 1980.

Haimes, Y.Y. "Hierarchical Holographic Modeling," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(9), 606-617, 1981.

Haimes, Y.Y. *Application of the Partitioned Multiobjective Risk Method to Dam Risk Analysis*, Report submitted to Oak Ridge National Laboratory, Oak Ridge, TN, 1986.

Haimes, Y.Y. "Total Risk Management," *Risk Analysis*, 11(2), 169-171, 1991.

Haimes, Y.Y., and V. Chankong. "Kuhn-Tucker Multipliers as Trade-offs in Multiobjective Decision-Making Analysis," *Automatica*, 15(1), 59-72, 1979.

Haimes, Y.Y., and C. Chittister. *An Acquisition Process for the Management of Risks of Cost Overrun and Time Delay Associated with Software Development,* Technical Report CMU/SEI-93-TR-28, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1993.

Haimes, Y.Y., and C. Chittister. "An Acquisition Process for the Management of Nontechnical Risks Associated with Software Development," *Acquisition Review Quarterly*, II(2), Defense Acquisition University, 121-154, Spring 1995.

Haimes, Y.Y. and W. Hall. *Multiobjective Optimization in Water Resources Systems: The Surrogate Worth Trade-off Method*, Elsevier, New York, 1975.

Haimes, Y.Y., and P.O. Karlsson. "Risk assessment of extreme events: application," *ASCE Journal of Water Resources Planning and Management* 115(3), 299-320, May 1989.

Haimes, Y.Y. and D. Li. "Multiobjective risk impact analysis method (MRIAM)," a chapter to appear in *Risk Management in a Multiobjective Decision Making Framework: Methodology and Applications,* in development, 1995.

Haimes, Y.Y., J.H. Lambert, and D. Li. "Risk of extreme events in a multiobjective framework," *Water Resources Bulletin* 28(1), Feb. 1992.

Haimes, Y.Y., R. Petrakian, P.O. Karlsson, and J. Mitsiopoulos. *Risk Analysis of Dam Failure and Extreme Floods: Application of the Partitioned Multiobjective Risk Method,* Final Report submitted to U.S. Army Corps of Engineers, Institute for Water Resources, Jan. 1988.

Haimes, Y.Y., K. Tarvainen, T. Shima, and J. Thadathil. *Hierarchical Multiobjective Analysis of Large-Scale Systems,* Hemisphere Publishing, New York, 1990.

Haimes, Y.Y., T. Barry, and J.H. Lambert (eds.) "When and how can you specify a probability distribution when you don't know much?," *Risk Analysis* 14(5), 661-701, 1994a.

Haimes, Y.Y., D. Li, J.H. Lambert, R.M. Schooff, S. Eisle, and C. Schneider. *Improving Risk Management for the Criminal Justice Information Services (FBI)*, Technical Report, Center for Risk Management of Engineering Systems, University of Virginia, Charlottesville, 1994b.

Hall, Arthur D., III. *Metasystems Methodology; A New Synthesis and Unification*, Pergamon Press, New York, 1989.

Heineman, G.T., J.E. Botsford, G. Caldiera, G.E. Kaiser, M.I. Kellner, N.H. Madhavji. "Emerging Technologies that Support a Software Process Life Cycle," *IBM Systems Journal*, 33(3), 501-529, 1994.

Herd, J.R. *Software Cost Estimation Study -- Study Results*, Technical Report, RADC-TR-77-220, Doty Associates, Rockville, MD, 1977.

Hudak J., B.H. Suh, D. Siewiorek, and Z. Segall. "Evaluation & Comparison of Fault-Tolerant Software Techniques," *IEEE Transactions on Reliability* 42(2), 190-204, June 1993.

Humphrey, W.S. *Managing the Software Process*, Addison-Wesley, Reading, Ma., 1989.

Humphrey, W.S., and M.I. Kellner. "Software Process Modeling: Principles of Entity Process Models," *Proceedings of the 11th International Conference on Software Engineering*, IEEE Computer Society Press, 175-188, May 1989.

Installe, M. *Decision Support Tools for Incentives Strategies Using Hierarchical Multicriteria Optimization*, Universite Catholique De Louvain, 1994.

Institute of Electrical and Electronic Engineers (IEEE). *IEEE Standard for Software Quality Metrics Methodology* (IEEE Std. 610.12-1990), IEEE, Inc., New York, 1990.

Jensen, R.W. "An Improved Macrolevel Software Development Resource Estimation Model," in *Proceedings of the 5th ISPA Conference*, 384-389, April 1983.

Johnson, B.W. *Design and Analysis of Fault Tolerant Digital Ssytems*, Addison-Wesley, Reading, MA, 1989.

Jones, C. *Programming Productivity*, McGraw-Hill, New York, 1986.

Jongen, H.T, and G.W. Weber. "On parametric nonlinear programming," *Annals of Operations Research*, 27, 253-284, 1990.

Kanoun K., M. Kaaniche, C. Beounes, J.C. Laprie, and J. Ariat. "Reliability Growth of Fault-Tolerant Software", *IEEE Trans. on Reliability* 42(2), 205-219, June 1993.

Kaplan, S., and B.J. Garrick. "On the Quantitative Definition of Risk," *Risk Analysis*, 1(1), 11-27, 1981.

Karlsson, P.O., and Y.Y. Haimes. "Risk-based analysis of extreme events," *Water Resources Research* 22(1), pp. 9-20, 1988.

Katz, R.W., A.H. Murphy, and R.L. Winkler. "Assessing the value of frost forecasts to orchardists: A dynamical decision-making approach," *Journal of Applied Meterology* 21(4), 518-531, 1982.

Kellner, M. I. "Software Process Modeling Support for Management Planning and Control," *Proceedings of the 1st International Conference on the Software Process: Manufacturing Complex Systems*, M. Dowson (ed.), IEEE Computer Society Press, 8-28, 1991.

Kellner, M.I., and G. Hansen. *Software Process Modeling*, Technical Report CMU/SEI-88-TR-9, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1988.

Kemerer, C.F. "An Empirical Validation of Software Cost Estimation Models," *Communications of the ACM*, 30(5), 416-429, Sept. 1987.

Kile, R.L. "A Method for Dynamic Cost Projection," *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Kitchenham, B.A., and N.R. Taylor. "Software Project Development Cost Estimation," *Journal of Systems and Software*, 5(4), 267-278, 1985.

Kitchenham, B.A., and J. G. Walker. "A quantitative approach to monitoring software develpment," *Software Engineering Journal*, 4(1), 2-13, Jan. 1989.

Knight J.C., Ammann, P.E. "Design Fault Tolerance", *Reliability Engineering and System Safety* 32, 25-49, 1991.

Krishnan, M.S. "Cost, Quality and User Satisfaction of Software Products: An Empirical Analysis," *Proceedings CASCON '93, Vol. I, Software Engineering*, National Research Council Canada, (A. Gawman, W.M. Gentleman, E. Kidd, P. Larson, J. Slonim, eds.), Toronto, Ontario, Canada, 400-411, Oct. 1993.

Krishnan, M.S., and M.I. Kellner. "An Empirical Relationship Between Software Cost and Product Quality," *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Krzysztofowicz, R. and D.R. Davis. "A methodology for evaluation of flood forecast-response systems, 1. Analyses and concepts, 2. Theory, 3. Case-studies," *Water Resources Research* 19(6), 1423-1454, 1983.

Kyle, R.A. *REVIC Software Cost Estimating Model User's Manual, Version 9.0*, Air Force Contract Management Division, Alberquerque, NM, Feb. 1991.

Lambert, J.H., D. Li, and Y.Y. Haimes. "Extreme values of monotonic functions and evaluation of catastrpohic flood loss," in *Extreme Value Theory and Applications*, J. Galambos, J. Lechner, and E. Simiu (eds.), U.S. Department of Commerce, National Institute of Standards and Technology, 1994.

Law, A.M., and W.D. Kelton. *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.

Leach, M. and Y.Y. Haimes. "Multiobjective risk-impact analysis method," *Risk Analysis* 7(2), 225-241, 1987.

Lederer, A.L., and J. Prasad. "Information Systems Software Cost Estimating: A Current Assessment," *Journal of Information Technology*, 8, 22-33, 1993.

Lenard, M.L. "Representing Models as Data," *Journal of Management Information Systems*, 2(4), 36-48, 1986.

Li, D., and Y.Y. Haimes. "The envelope approach for multiobjective optimization problems," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-17, 1026-1038, 1987.

Li, D., and Y.Y. Haimes. "Decomposition techniques in multiobjective discrete-time dynamic problems," in *Advances in Control and Dynamic Systems*, (C.T. Leondes, ed.), Academic Press, New York, Vol. 28, 1988.

Li, D., Y.Y. Haimes, E. Stakhiv, and D. Moser. "Optimal flood warning threshold and a case study in Milton, Pennsylvania," in *Risk-Based Decision Making in Water Resources V*, Y.Y. Haimes, D. Moser, and E. Staihive (eds.), American Society of Civil Engineering, New York, 260-283, 1992.

Liang, T.-P. "Integrating Model Management with Data Management in Decision Support Systems," *Decision Support Systems*, 1(3), 221-232, 1985.

Londeix, B. *Cost Estimation for Software Development*, Addison-Wesley, Reading, MA, 1987.

Lowrance, W.W. *Of Acceptable Risk: Science and Determination of Safety*, William Kaufmann, Inc., Los Altos, California, 1976.

Luce, R.D., and H. Raiffa. *Games and Decision*, Wiley, New York, 1957.

Lyu, M.R., and Y.T. He. "Improving the N-version Programming Process Through the Evolution of a Design Paradigm," *IEEE Transactions on Reliability*, 42(2), 179-189, June 1993.

Macko, D., and Y.Y. Haimes. "Overlapping Coordination of Hierarchical Structures," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-8(10), 1978.

Masters, T.F., II. "An Overview of Software Cost Estimating at the NSA," *Journal of Parametrics*, 5(1), 72-84, Mar. 1987.

Matson, J.E., B.E. Barrett, and J.M. Mellichamp. "Software development cost estimation using function points," *IEEE Transactions on Software Engineering* 20(4), 275-287, Apr. 1994.

McFarlan, F.W. "Portfolio Approach to Information Systems," *Harvard Business Review*, 142-150, Sept.-Oct. 1981.

McRitchie, K. *Using SEER-SEM to Estimate Commercial Off the Shelf Integration*, paper presented to the Southern California chapter of ISPA, Feb. 1995, quoted in "Cost Factors for COTS Integration," R.D. Stutzke, *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Miller, L.W., and N. Katz. "Model Management Systems to Support Policy Analysis," *Decision Support Systems*, 2(1), 55-63, Jan. 1986.

Mills, E. *Software Metrics*, Technical Report SEI-CM-12-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1988.

Mitra, S., and A. Dutta. "Integrating Optimization Models and Human Expertise in Decision-Support Tools," *Expert Systems with Applications*, 7(1), 93-107, Jan. 1994.

Mohanty, S.N. "Software Cost Estimation: Present and Future," *Software - Practice and Experience*, 11, 103-121, 1981.

Muhanna, W.A. "SYMMS -- A Model Management System that Supports Model Reuse, Sharing, and Integration," *European Journal of Operational Research*, 72(2), 214-242, Jan. 1994.

Muhanna, W.A., and R.A. Pick. "Metamodeling Concepts and Tools for Model Management," *Management Science*, 40(9), 1093-1123, Sept. 1994.

Musa, J.D., A. Iannino, and K. Okumoto. *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1990.

Navlakha, J.K. "Choosing a Software Cost Estimation Model for Your Organization: A Case Study," *Information and Management*, 18, 255-261, May 1990.

Neirenberg, G. *The Art of Negotiating*, Negotiation Institute, New York, 1978.

Nejmeh, B.A. "Designs on CASE," *Unix Review* 6(11), 1990.

Nelson, E.A. *Management Handbook for the Estimation of Computer Programming Costs*, AD-A648750, Systems Development Corp, Oct. 1966.

Neufelder, A.M. *Ensuring Software Reliability*, Dekker, New York, 1993.

Neumann, P.G. "Risks to the Public in Computers and Related Systems," *ACM Software Engineering Notes*, 5-18, April 1988.

Nijkamp, P., and N. Rietveld. "Multi-objective multi-level policy model: an application to regional and environmental planning," *European Economic Review*, 15, 63-89, 1981.

Nijkamp, P., and J. Spronk. "Interactive multiple goal programming: an evaluation and some results," in *Multiple Criteria Decision Making Theory and Applications -- Hagen/Sonigswinter*, West Germany 1979 (G. Fandel and T. Gal, eds.), Springer-Verlag, Berlin, 278-293, 1980.

Norden, P.V. "Useful Tools for Project Management," from *Operations Research in Research and Development*, B.V. Dean (ed.), Wiley, New York, 1963.

Pages, E.R. Testimony before a Joint Hearing of the Senate Committee on Armed Services and Senate Committee on Governmental Affairs, Washington D.C., March 10, 1994.

Palisade Corporation. *@RISK: Risk Analysis and Simulation Add-in for Microsoft Excel or Lotus 1-2-3,* Palisade Corporation, Newfield, NY, 1995.

Paulk, M.C., B. Curtis, M.B. Chrissis, E. Averill, J.Bamberger, T. Kasse, M. Konrad, J. Perdue, C. Weber, J. Withey. *The Capability Maturity Model for Software, 1992 SEI Technical Review*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1992.

Paulk, M.C., C.V. Weber, S.M. Garcia, M.B. Chrissis, M. Bush. *Key Practices of the Capability Maturity Model, Version 1.1*, Technical Report CMU/SEI-93-TR-25, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1993.

Perlis, A.J., F. Sayward, and M. Shaw. *Software Metrics : An Analysis and Evaluation*, MIT Press, Cambridge, Mass, 1981.

Prentice, R.L., B.J. Williams, and A.V. Peterson. "On the Regression Analysis of Multivariate Failure Time Data," *Biometrika*, 68, 373-379, 1981.

Pressman, R.S. *Software Engineering: A Practitioner's Approach*, 2nd ed., McGraw-Hill, New York, 1987.

Price Systems. "The Central Equations of the Price-S Software Cost Model," *Proceedings of the Fourth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1988.

Przeminiecki, J. *Acquisition of Defense Systems*, American Institute of Aeronautics and Astronautics, Washington D.C., 1993.

Putnam, L.H. "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, 345-361, July 1978.

Putnam, L.H., and A. Fitzsimmons. "Estimating Software Costs," *Datamation*, Sept. - Nov. 1979.

Putnam, L.H., and W. Myers. *Measures for Excellence: Reliable Software On Time Within Budget,* Yourdon Press, Prentice Hall, Englewood Cliffs, NJ, 1992.

Quantitative Software Management (QSM) Inc. *Slim User's Manual*, QSM, McLean, VA, 1987.

Raiffa, H. *The Art and Science of Negotiation*, Belknap Press of Harvard University Press, Cambridge, MA, 1982.

Reid, J.G. *Linear System Fundamentals: Continuous and Discrete, Classic and Modern*, McGraw-Hill, New York, 1983.

Rifkin, S. *Level 5 CMM Companies,* electronic mail communication, Master Systems Inc., George Washington University, Washington D.C., Jan.1995.

Romei, S., Y.Y. Haimes, and D. Li. "Exact determination and sensitivity analysis of a risk measure of extreme events," *Information and Decision Technologies* 18, 265-282, 1992.

Ross, S. *Introduction to Probability Models*, Academic Press, Boston, 1989.

Rothfeder, J. "It's late, costly, incompetent–But try firing a Computer System," *Business Week*, 164-165, November 7, 1988.

Rowe, W.D. *An Anatomy of Risk*, John Wiley & Sons, New York, 1977.

Royce, W.W. "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings WESCON*, Aug. 1970.

Rozum, J.A. *Software Measurement Concepts for Acquisition Program Managers*, Technical Report CMU/SEI-92-TR-11, Software Engineering Institute, Carnegie Mellon University, Pitsburgh, PA, 1992.

Sage, A.P. *Methodology for Large-Scale Systems*, McGraw-Hill, New York, 1977.

Sage, A. *Software Systems Engineering*, Wiley, New York, 1995.

Schneidewind, N.F. "Software Reliability Model with Optimal Selection of Failure," *IEEE Transactions on Software Engineering*, 19(11), Nov. 1993.

Schulmeyer, G.G., and J.I. McManus. *Total Quality Management for Software*, Van Nostrand Reinhold, New York, 1992.

Science, Space, and Technology Committee (SST). *Bugs in the Program: Problems in Federal Government Computer Software Development and Regulation*, Subcommittee on Investigations and Oversight, Committee on Science, Space, and Technology, U.S. House of Representatives, Washington D.C., Sept. 1989.

Scientific Advisory Board (SAB). *Information Architectires that Enhance Operational Capability in Peacetime and Wartime*, HQ AF/SB, Department of the Air Force, Washington D.C., Feb. 1994.

Shaw, M.J., P.L. Tu, and P. De. "Applying Machine Learning to Model Management in Decision Support Systems," *Decision Support Systems*, 4, 285-305, 1988.

Shepperd, M. *Foundations of Software Measurement*, Prentice Hall, Englewood Cliffs, NJ, 1995.

Sherer, S.W., and J. Cooper. *Software Acquisition Maturity Model (SAMM) Draft Version 4.0*, Special Report, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Sept. 1994.

Shima, T., and Y.Y. Haimes. "The Convergence Properties of Hierarchical Overlapping Coordination," *IEEE Transactions on Systems, Man, and Cybernetics*, 14(1), 1984.

Sobol, I.M. "A global search for multicriteria problems," *Multiple Criteria Decision Making: Theory and applications in Business, Industry, and Government, Proceedings of the Ninth International Conference*, (A. Goicoechea, L. Duckestein, and S. Zionts, eds.), Springer-Verlag, New York, 401-412, 1992.

Software Technology Support Center (STSC). *Project Management Technology Report*, technical report, STSC, Hill AFB, UT, Dec. 1993.

Sprague, R.H., and E.D. Carlson. *Building Effective Decision Support Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

Stackelberg, H. von, *The Theory of the Market Economy*, Oxford University Press, London, 1952.

Stohr, E., and M. Tanniru. "A Database for Operations Research Models," *International Journal of Policy Analysis and Information Systems*, 4(1), 105-121, Jan. 1980.

Stutzke, R.D. "Cost Factors for COTS Integration," *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Tai, A.T., J.F. Meyer, and A. Algirdas. "Performability Enhancement of Fault Tolerant Software", *IEEE Transactions on Reliability* 42(2), June 1993.

Tarvainen, K. and Y.Y. Haimes. "Coordination of Hierarchical Multiobjective Systems: Theory and Methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-12(6), 1982.

Vaidya, N.H. and D.K. Pradhan.``Fault-Tolerant Design Strategies for High-Reliability and Safety," *IEEE Transactions On Computers* 42(10), 1195-1206, Oct. 1993.

Vicinanza, S.S., T. Mukhopadhyay, and M.J. Prietula. "Software Effort Estimation: An Exploratory Study of Expert Performance," *Information Systems Research*, 2(4), 243-262, 1991.

Von Neumann, J., and O. Morgenstern. *Theory of Games and Economic Behavior*, Princeton University Press, Princeton, 1944.

Wallenius, J. "Comparative evaluation of some interactive approaches to multicriterion optimization," *Management Science*, 21, 1387-1396, 1975.

Waltson, C.E., and C.P. Felix. "A Method of Programming Measurement and Estimation," *IBM Systems Journal*, 16(1), 54-73, 1977.

White, D.J. *Optimality and Efficiency*, Wiley, New York, 1982.

Wolverton, W.R. "Airborne Systems Software Acquisition Engineering Guidebook: Software Cost Analysis and Estimating," U.S. Air Force ASD/EN, Wright Patterson AFB, OH, Feb. 1980.

Zhu, W., and B. Lowther. "COCOMO Estimation templates for Excel," *IEEE Software*, pp. 115-119, Oct. 1993.

Appendix A
A COCOMO Tutorial

The COnstructive COst MOdel (COCOMO) is widely recognized within the software community as the predominant software estimation methodology. In light of COCOMO's preeminence, the approaches developed through this research are deployed to this model. This Appendix presents an overview of the parameters and methodology of the model -- the three models that constitute COCOMO -- and a brief discussion of recent advances concerning the model.

A.1 An Overview of COCOMO

While there are many software cost estimation models, each having its own followers and advocates, no one model has been shown to provide a definitive estimation solution. Although this is true, due to its widespread use, open publication of its methodology, extensive application to a wide variety of software, and adaptations for modern software practices, COCOMO has become the de facto standard for software cost estimation [Charette 1989].

Originally developed in the early 1980's, the complete COCOMO and associated data base from which the model was developed appear in [Boehm 1981]. COCOMO uses development effort equations to estimate the total man-months (MM) of development effort required to complete a project. An initial project cost estimate can then be derived by multiplying MM by a dollar-per-month cost multiplier. COCOMO also includes an equation for estimating development time (t_D) in months as a function of MM. The effort equations require project size estimates, measured in thousands of delivered source instructions (KDSI), and estimates of other key cost drivers.

KDSI offers a more specific definition of the often-misinterpreted KLOC. *Delivered* is meant to exclude nondelivered support software. *Source instruction* "includes all program instructions created by project personnel, and processed into machine code by some combination of preprocessors, compilers, and assemblers" [Boehm 1981]. KDSI *excludes* comment statements and unmodified utilities, while including job control language, format

statements, data declarations, and instructions. For the purposes of this paper, the terms KLOC and KDSI are used interchangeably.

COCOMO considers three software project development modes that define three types of software development environments: organic, semidetached, and embedded. The *organic mode* involves development by relatively small teams in a highly familiar, in-house environment. An organic-mode project is similar to previously developed products, is relatively small, and requires little innovation. An accounting system is an example of an organic-mode project.

An *embedded-mode* project is typified by tight, inflexible constraints and interface requirements, and requires a great deal of innovation. The project is developed "under a strongly coupled complex of hardware, software, regulations, and operational procedures." [Boehm 1981]. Real-time systems with critical timing constraints and customized hardware are generally embedded-mode projects.

The *semidetached-mode* project is a combination of the organic and embedded modes, lying somewhere in-between the other two modes in terms of complexity, size, and required innovation.

COCOMO consists of three models of increasing complexity: Basic COCOMO, Intermediate COCOMO, and Detailed COCOMO. The primary distinction among the models is the detail and number of model parameters. While the less-detailed Basic COCOMO is usually appropriate for quick, rough estimation, the higher-detailed models may not necessarily produce more accurate cost projections unless accurate estimates of the additional model parameters are obtainable. The requirement to estimate a greater number of model parameters introduces additional uncertainty into the more-detailed models. The following sections provide mathematical descriptions of the three software cost estimation models that constitute COCOMO.

A.2 Basic COCOMO

The Basic COCOMO's estimate of required development effort is based entirely upon the user's estimate of project size and correct determination of the development mode. The

essential effort equation of the Basic COCOMO model is a nonlinear function of the estimated project size

$$MM = a(KLOC)^b \tag{A.1}$$

where the parameter vector <*a*,*b*> takes on differing values according to the development mode of the project. The value MM that is produced from the effort equation is used in the schedule equation to estimate development time t_D (in months)

$$t_D = c(MM)^d. \tag{A.2}$$

Table A.1 lists the Basic COCOMO development effort and development time equations.

Table A.1 Basic COCOMO Equations [Boehm 1981]

Mode	Development Effort	Development Time
Organic	$MM = 2.4(KLOC)^{1.05}$	$t_D = 2.5(MM)^{0.38}$
Semidetached	$MM = 3.0(KLOC)^{1.12}$	$t_D = 2.5(MM)^{0.35}$
Embedded	$MM = 3.6(KLOC)^{1.20}$	$t_D = 2.5(MM)^{0.32}$

The total development effort and total development time are distributed over the life cycle phases according to distribution percentages determined from analysis of completed projects. The differences in the software development activities among the three modes produce different estimated phase distributions of effort and schedule. For example, embedded-mode projects consume more effort in the integration and test phase and proportionally less effort in the code and unit test phase [Boehm 1981].

The effort and schedule results of the Basic COCOMO can be used to evaluate pertinent software development measures. Average personnel staffing requirements can be estimated as

$$FSP = \frac{MM}{t_D} \tag{A.3}$$

where *FSP* stands for full-time-equivalent software personnel, a measure of the equivalent number of people working on the project at a given time. A measure of average productivity (KLOC per man-month of effort) can be determined by

$$\text{Productivity} = \frac{KLOC}{MM}. \tag{A.4}$$

Use of the Basic COCOMO model is intended for initial estimation, providing quick, rough order-of-magnitude estimates of a software project's effort and cost. The model requires estimation of a single parameter, which makes implementation of the model quite easy. Obviously, the importance of accurately estimating the KLOC parameter cannot be overstated. In accuracy tests using ex-post (actual) KLOC data, the reported accuracy of the Basic COCOMO model is generally within a factor of 2 of the actual results 60% of the time and within a factor of 1.3 of the actual results 29% of the time [Barrow et al. 1993], [Boehm 1981].

A.3 Intermediate COCOMO

The Intermediate COCOMO model improves upon the Basic COCOMO model's approach by including the effect of 15 "cost drivers" or effort multipliers The Intermediate COCOMO model begins by assessing a nominal effort estimate, using equations of the same form as those used in the Basic COCOMO:

$$MM_{NOM} = a(\text{KLOC})^b. \tag{A.5}$$

The user's assessment of the software project's attributes, relative to the 15 cost drivers, provides the means for "fine-tuning" the nominal effort estimate. These cost drivers are grouped in four categories: software product attributes, computer attributes, personnel attributes, and project attributes (Table A.2).

Table A.2 Intermediate COCOMO Effort Multipliers [Boehm 1981]

Product Attributes	Project Attributes
Required Software Reliability	Modern Programming Practices
Data Base Size	Use of Software Tools
Product Complexity	Required Development Schedule
Computer Attributes	**Personnel Attributes**
Execution Time Constraints	Analyst Capability
Main Storage Constraints	Applications Experience
Virtual Machine Volatility	Programmer Capability
Computer Turnaround Time	Virtual Machine Experience
	Programming Language Experience

The user must rate each attribute on a scale from Very Low to Extra High, where each rating has an associated numerical score that represents an effort adjustment multiplier.

Nominal ratings have an adjustment multiplier of 1.0, while the numerical range of each multiplier is from 0.0 to 2.0. These 15 numerical scores are combined to produce an overall effort adjustment factor (EAF), defined as the product of all attribute scores

$$EAF = \prod_{i=1}^{15} e_i, \tag{A.6}$$

where e_i is the effort multiplier score of a particular attribute. The development effort equation of the Intermediate COCOMO then incorporates the effect of the EAF multiplier Eq. (A.6) on the nominal development effort Eq. (A.5):

$$MM = (EAF)MM_{NOM} = (EAF)[a(KLOC)^b]. \tag{A.7}$$

The Intermediate COCOMO effort equations for the three development modes are listed in Table A.3; the development time equations are the same as those of Basic COCOMO. Distribution of time and effort among life cycle phases, as well as productivity and staffing measures, are developed as with Basic COCOMO.

Table A.3 Intermediate COCOMO Equations [Boehm 1981]

Mode	Development Effort	Development Time
Organic	$MM = (EAF)[3.2(KLOC)^{1.05}]$	$t_D = 2.5(MM)^{0.38}$
Semidetached	$MM = (EAF)[3.0(KLOC)^{1.12}]$	$t_D = 2.5(MM)^{0.35}$
Embedded	$MM = (EAF)[2.8(KLOC)^{1.20}]$	$t_D = 2.5(MM)^{0.32}$

The consideration of the effort adjustment factors contributes to the Intermediate COCOMO's improved estimation capabilities. Again, when considering ex-post KLOC and effort multiplier data, the Intermediate COCOMO's estimation accuracy is reported to be usually within 20% of the actual results 68% of the time [Barrow et al. 1993], [Boehm 1981]. Obviously, the user's ability to accurately determine the development mode, estimate the project size, and evaluate all 15 cost multipliers is the key determinant affecting the accuracy of the model's results.

The Intermediate COCOMO model has been the most widely implemented of the COCOMO models and, therefore, the most widely implemented of all software estimation models.

A.3.1 Example Software Estimation Using Intermediate COCOMO

To demonstrate the Intermediate COCOMO model, we consider the following example software development scenario. A 32-KLOC, semidetached-mode software project is to be developed. The values of the model parameters and the cost multiplier values of the attributes characterizing the development effort are listed in Table A.4.

Table A.4 Intermediate COCOMO example problem -- model values

Variable		
KLOC =	32	
Mode =	Semidetached, hence from Table A.3	
	a = 3.00	
	b = 1.12	
	c = 2.50	
	d = 0.35	
Attribute	**Rating**	**Adjustment Factor (e)**
Reliability	Nominal	1.00
Data Base Size	Low	0.94
Complexity	Very High	1.30
Execution Time	High	1.11
Storage	High	1.06
Virtual Machine Volatility	Nominal	1.00
Turnaround Time	Nominal	1.00
Analyst Capability	High	0.86
Applications Experience	Nominal	1.00
Programmer Capability	High	0.86
Virtual Machine Experience	Low	1.10
Programming Language	Nominal	1.00
Programming Practices	High	0.91
Software Tools	Low	1.10
Development Schedule	Nominal	1.00
	EAF, Eq. (A.6) =	1.171

Applying the above values to Eq. (A.5), the nominal development effort is

$$MM_{NOM} = 3.0(32)^{1.12} = 145.51 \text{ man-months.}$$

Adjusting the nominal effort to consider the project-specific attributes, leads to the adjusted development effort requirement Eq. (A.7)

$$MM = EAF(MM_{NOM}) = (1.171)(145.51) = 170.39 \text{ man-months.}$$

The development time is given by Eq. (A.2)

$$t_D = c(MM)^d = 2.5(170)^{0.35} = 15.087 \text{ months.}$$

Hence, the initial estimate for project development effort and schedule is 170 man-months of effort over a 15-month development period. The required average staffing and productivity measures can be calculated by Eqs. (A.3) and (A.4):

$$FSP = \frac{MM}{t_D} = (170)/(15) = 11.33 \text{ persons,}$$

$$\text{Productivity} = \frac{KLOC}{MM} = (32)/15) = 2.133 \text{ KLOC per man-month.}$$

Additionally, the effort, schedule, and average staffing analysis can conducted for each phase of the development life cycle using percentage distributions [Boehm 1981]. The phase distribution values and the overall development resource requirements for this example are summarized in Table A.5. Such information is useful for developing staffing plans, and for determining and then monitoring project progress.

Table A.5 Intermediate COCOMO Example - Phase Distribution of Resources

Life Cycle Phase	Effort (man-months)	Schedule (months)	Avg. Staffing (FSP)
Plans & Requirements	11.90	3.00	4.00
Development			
Design	28.90	3.90	7.4
Detailed Design	42.50	3.10	13.7
Code & Test	56.10	4.10	13.7
Integration & Test	42.50	3.90	10.9
Total Development	170.00	15.00	

A.4 Detailed COCOMO

The Detailed COCOMO model builds upon the Intermediate model by considering the effect of changing effort multipliers as the software development effort progresses through the life cycle. With the Detailed COCOMO, the project is divided into at least four phases (Requirements and Product Design, Detailed Design, Code and Unit Test, Integrate and

Test) and the 15 cost drivers are estimated separately for each phase, rather than for the project as a whole.

The effort equations of the Detailed COCOMO are essentially the same as those of the Intermediate COCOMO, however the total MM of development effort is now accounted for by considering the effort estimates of each phase. When considering four development phases, the Detailed COCOMO requires the user's estimate of development mode, KLOC, and 60 cost drivers (15 in each of four phases). This additional detail, however, has not resulted in a significant improvement in cost estimation accuracy. The Detailed COCOMO has been shown to produce only modest improvement over the Intermediate COCOMO, estimating within 20% of project actuals 70% of the time [Boehm 1981].

A.5 Ada COCOMO

With the directive that the programming language Ada be the language of choice for new Department of Defense software systems, a COCOMO model especially tailored for Ada projects was needed. Many factors that influence the development of Ada software were not considered by the standard COCOMO estimation equations. An initial Ada COCOMO was developed for estimating software products developed in the Ada programming language [Boehm and Royce 1987]. Ada COCOMO includes additional effort multipliers, consideration of a phase distribution unique to Ada projects, and revised model coefficient values <*a,b, c, d*>. Since initial development, the Ada COCOMO has been tested and refined. Additional applications and adjustments are part of an ongoing effort[Boehm and Royce 1989].

A.6 COCOMO 2.0

Software development trends towards reuse, re-engineering, and commercial off-the-shelf (COTS) packages, object orientation, non-sequential process models, rapid development and prototype approaches motivate a recent effort for updating COCOMO [Boehm et al. 1995]. While still in the design stage, this model (designated COCOMO 2.0) includes consideration of spiral and prototype development paradigms, software reuse and re-engineering, COTS, and other current software engineering practices, technologies, and environments. COCOMO 2.0 will allow a range of software sizing options: FP, KLOC,

and object points [Boehm 1995]. Furthermore, capabilities for integrating reused and COTS software and re-engineering and conversion efforts are to be included [Boehm 1995]. A comprehensive set of recently completed software projects will be used to update the effort-multiplier cost drivers to reflect the most current software development effores.

A.7 Summary

COCOMO is the most widely used, most openly published software estimation model. The three increasingly-complex COCOMO models provide a range of software estimation sophistication. The outputs of the COCOMO models, especially the Intermediate and Detailed models, have been shown to be reliable *if the number of lines of code is reasonably well known and the multipliers can be chosen correctly.* However, even with perfect input data, their projected accuracy is only within 20 percent of actual costs 70 percent of the time.

These reported accuracy levels define the upper limit of accuracy for software cost estimation, as these have been evaluated using actual, ex-post data. This can pose a serious risk to a user of the model if the projected accuracy -- the uncertainty of the estimate -- is ignored. The accuracy falls off in practice when users are faced with estimating parameter values in an uncertain environment, disregard good estimation practices, or use the cost estimation exercise to justify a previously predicted result. Explicitly accounting for the uncertainty associated with the COCOMO's effort estimation is a key contribution of this research.

A.8 References for Appendix A

Banker, R., H. Chang, and C. Kemerer. "Evidence on Economies of Scale in Software Development," *Information and Software Technology*, 1994.

Barrow, D., S. Nilson, and D. Timberlake. *Software Estimation Technology Report*, Software Technology Support Center (SCTC), Hill AFB, UT, 1993.

Boehm, B.W. "COCOMO 2.0: Recent Developments," *Proceedings of the Tenth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Oct. 1995.

Boehm, B.W., B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Shelby. "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0," to appear in *Annals of Software Engineering Special Volume on Software Process and Product Measurement*, J.D. Arthur and S.M. Henry (eds.), J.C. Baltzer AG Science Publishers, Amsterdam, 1995.

Boehm, B.W., and W. Royce. "Ada COCOMO and the Ada Process Model," *Proceedings, Fifth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1989.

Boehm, B.W., and W. Royce. "Ada COCOMO: TRW IOC Version," *Proceedings, Third COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Nov. 1987.

Boehm, B.W. *Software Engineering Economics*, Prentice-Hall, New Jersey, 1981.

Charette, R.N. *Software Engineering Risk Analysis and Management*, McGraw Hill, New York, 1989.

Appendix B
Software Estimation Tools

This Appendix provides a summary of the major software estimation tools available from private vendors and public agencies. Much of the information came from the product vendors themselves, and other information came from software estimation tool reviews such as [Barrow et al. 1993].

The COCOMO is widely used as a baseline for many estimation tools because it is considered an "open model" since all details are well documented and published. For this reason, and that there are so many computerized estimation tools available that use COCOMO, a separate section on these tools is provided (B.1). Function Point analysis is increasingly gaining acceptance as a size-estimation methodology, with particular application to Information Systems. Section B.2 discusses tools based on Function Point analysis. Finally, Section B.3 describes additional software estimation methdologies that are based on methodologies other than COCOMO or Function Points.

B.1 COCOMO-Based Software Estimation Tools

This Section discusses COCOMO-based estimation tools.

B.1.1 CB COCOMO

Crystal Ball (CB) COCOMO is distributed by Decisioneering Incorporated. This tool estimates time and cost of software development projects, and allows entering of actual project data at various phases in the software life cycle. This tool requires Decisioneering's forecasting and risk analysis tool, Crystal Ball, and operates on Macintosh and DOS systems. Decisioneering Inc., Boulder CO 80301. (303) 292-2291.

B.1.2 COCOMOID

COCOMOID is distributed by the Air Force Cost Center (via electronic bulletin board) and the Society for Cost and Economic Analysis (SCEA). COCOMOID is a complete

COCOMO implementation supporting all known published COCOMO specifications. COCOMO models used include Basic, Intermediate, Detailed, and Maintenance models. It also includes Enhanced Ada, Ada Process, and Incremental Development models. This tool operates on PC compatible systems. Air Force Cost Center, Wright-Patterson AFB, OH 45433. (513) 257-3927.

B.1.3 COCOMO1

COCOMO1 was developed by Level Five Research, Inc. and is marketed by Solar Powered Emergency Communications Systems (SPECS), Inc. It is an expert systembased software tool that estimates cost and time requirements for software projects. Through a series of questions COCOMO1 assists in determining the proper COCOMO cost model, mode, and effort coefficients. This tool uses fifteen development and maintenance cost drivers and applies formulas from the COCOMO model to these factors. COCOMO1 runs on all PC compatible systems. Specs Inc., Junction City, OR 97448. (503) 998-8729.

B.1.4 CoCoPro

CoCoPro is distributed by Iconix Software Engineering, Inc. It estimates resources needed to complete software development projects using the COCOMO model. This tool uses exponential functions and attributes to calculate development costs. Inputs allowed include personnel experience and capabilities, project complexity, product factors, and hardware limitations. CoCoPro operates on a Macintosh. Iconix Software Engineering, Inc., Santa Monica, CA 90405. (310) 458-0092.

B.1.5 COSTAR

COSTAR is an interactive software cost estimation tool marketed by Softstar Systems. It is a full implementation of the detailed COCOMO model and offers side-by-side comparisons of several alternative estimates. This tool also provides automatic recalculation and display of results, and uses definable cost drivers. COSTAR version 3.0 includes support of Ada COCOMO. This tool also uses Function Point analysis for software size estimations. It is available for both PC compatible and VAX computer systems. Softstar Systems, Amherst, NH 03031. (603) 672-0987.

B.1.6 COSTMODL

COSTMODL was developed by the Software Technology Branch, Spacecraft Software Division, NASA/Johnson Space Center [NASA/JSC 1991], and provides estimates for effort, cost, and schedule. It implements all the COCOMO models except the detailed model. In addition, it includes a simplified linear model using productivity data from completed NASA projects.

COSTMODL is presently used at over 100 government, military, and contractor sites, as well as NATO headquarters, the British Ministry of Defense, and several universities in the United States and England. It has been selected as the standard cost estimating tool for NASA's Space Station Freedom Software Support Environment.

COSTMODL contains five different models for estimating non-Ada and Ada products, and products which are to be delivered as a series of incremental development phases. All of the parameters defining each of the models are accessible to the user. Basic estimating equations can be calibrated to the user's software development environment and type of product. Also, the set of factors which influence software development costs can be redefined.

Given the data describing the software development productivity experience for a user's organization, COSTMODL will automatically compute the coefficients and exponents which will provide the most meaningful estimates for new products to be developed within that organization. It also contains an extensive set of linked, context-sensitive help displays and demonstration files designed to quickly familiarize the new user with its operation.

This tool is free to both employees and contractors of NASA, as well as other government agencies, and has been submitted to NASA's Computer Software Management and Information Center (COSMIC) for distribution into the private sector. COSTMODL currently runs on PC compatible systems. NASA/JSC Software Technology Branch, Houston, TX 77058. (713) 483-9092.

B.1.7 GECOMO Plus

GECOMO Plus is marketed by GEC-Marconi Software Systems. It is a special enhancement of the COCOMO model and uses 17 cost drivers. It provides cost estimations

for both non-Ada and Ada projects. GEC-Marconi markets a companion tool for size estimation called SIZE Plus. Both GECOMO Plus and SIZE Plus are X-Windows/OSF Motif compatible and are available for both the UNIX and VMS operating systems. GEC-Marconi Software Systems, Reston, VA 22090. (703) 648-1551.

B.1.8 GHL COCOMO

GHL COCOMO is marketed by GHL Associates, Inc. and features three levels of detail: multiproject, data retention, and sensitivity analysis. This tool also allows for "what-if" scenarios. It operates on PC compatible systems. GHL Associates, Inc., Haverford, PA 19041. (215) 896-7307

B.1.9 REVIC

The Revised Enhanced Version of Intermediate COCOMO (REVIC) was developed by Hughes Aerospace. The Air Force Contract Management Division, Air Force System Command, Kirtland Air Force Base, New Mexico, sponsored the development for use by its contract administrator [Kyle 1991]. The main difference between REVIC and COCOMO is the coefficients used in the effort equations. REVIC changed the coefficients based on using a database of recently completed DOD projects. It also uses a different method of distributing effort and schedule to each phase of product development, and applies an automatic calculation of standard deviation for risk assessment

REVIC provides a single-weighted "average" distribution for effort and schedule along with the ability to let the user vary the percentages in the system engineering and development test and evaluation phases. REVIC employs a different Ada model than Ada COCOMO. The REVIC model has also been enhanced by using a Program Evaluation and Review Technique (PERT) statistical method for determining the lines of code to be developed.

In addition to providing estimates for cost, manpower and schedule, the program creates estimates for typical DOD-STD-2167A documentation sizing and long term software maintenance (with planned modification to meet MIL-STD-498 requirements). REVIC's schedule estimates are often considered lengthy because it assumes that a project's documentation and reviews comply with the full requirements of DOD-STD-2167A.

REVIC operates on PC compatible systems. Air Force Cost Agency, Arlington, VA 22202. (703) 746-5865.

B.1.10 SECOMO

Software Engineering Cost Model (SECOMO), an implementation of COCOMO, is available at no cost from IIT Research Institute in Rome, NY. Enhancements include an improved user interface, online help, and an expanded user's manual. Version 7.0 will operate on a PC compatible system or a VAX/VMS 5.2 or later operating system. IIT Research Institute, Rome, NY 13440. (315) 336-2359.

B.1.11 SWAN

The Software Analysis (SWAN) cost model was developed by IIT Research Institute for the U.S. Army Program Manager for Training Devices (PMTRADE) in the Ada programming language. SWAN is available at no cost to government agencies and associated contractors

This tool supports the intermediate version of COCOMO, including Ada COCOMO with full three-level software hierarchy support. SWAN utilizes FPA techniques to determine software size estimates. It provides estimates for software development from requirements analysis through integration and test, as well as estimates for up to 5 years of maintenance. SWAN runs on PC compatible systems under MS-DOS 3 1 or later. IIT Research Institute, Rome, NY 13440. (315) 336-2359.

B.2 Function Point-Based Software Estimation Tools

This Section contains a discussion of software estimation tools based on Function Point analysis.

B.2.1 ASSET-R

ASSET-R is a function point sizing tool developed to estimate the size of data procesing, real-time, and scientific software systems which is marketed by Reifer Consultatnts, Inc. It utilizes a knowledge-based system which extends the theory of function points into

scientific and real-time systems by considering issues like concurrence, synchronization, and reuse in its mathematical formulation. The formulas use as many as nine parameters to develop function point counts. It operates on PC compatible systems. Reifer Consultants, Torrance, CA 90510. (310) 373-8728.

B.2.2 CA-FPXpert

CA-FPXpert is distributed by Computer Associates International, Inc. It uses FP analysis for size estimation of Information System type software projects. It includes an on-line tutor to help the function point counting process. CA-FPXpert works in conjunction with CA_ESTIMACS to provide software size estimation input and operates on PC compatible systems. Computer Associates International, Inc., Calverton, MD 20705. (301) 937-1133.

B.2.3 CHECKPOINT

CHECKPOINT is a software estimation tools distributed by Software Productivity Research (SPR). It is a knowledge-based software estimation tool that has largely superseded SPQR/20. It's algorithms are derived from measurements of more than 4200 software projects, and it is applicable to all phases of the software development life cycle. It applies to all types of programs and incorporates Function Points or Feature Points to calculate the size of a software product. Feature points are SPR's method of measuring functionality. Software Products Research, Inc., Burlington, MA 01803. (617) 273-0140.

B.2.4 MicroMan ESTI-MATE

MicroMan ESTI-MATE is an estimating and planning tool for Information Systems oriented projects. It uses FP methodologies, and is distributed by POC-IT Management Services, Inc. MicroMan ESTI-MATE provides a breakdown of the hours required for all phases, activities, and tasks that make up a project. It is fully integrated with the MicroMan II Project and Staff Management System tool used for scheduling, tracking, and reporting. MicroMan ESTI-MATE operates on PC compatible systems. POC-IT Management Services, Inc., Santa Monica, CA 90401. (301) 393-4552.

B.2.5 PROJECT BRIDGE

PROJECT BRIDGE Planning and Estimating System is marketed by Applied Business Technology Corporation. It is a knowledge-based tool used for profiling, estimating, and planning projects in a software engineering environment. It allows users to produce estimates based on Function Points or an organization's own estimating factors. This tool integrates with the Project Workbench project management tool for Information Systems projects. PROJECT BRIDGE operates on PC compatible systems. Applied Business Technology Corp., New York, NY 10013-3992. (800) 444-0724.

B.2.6 SIZE Plus

SIZE Plus is marketed by GEC-Marconi Software Systems. This tool uses FP analysis to estimation the size of the software project. It supports both data processing and real-time applications. SIZE Plus provides five different methods to perform FP analysis. Three of these are oriented towards Information System applications and the other two are used for real-time or embedded software applications. The tool is available for UNIX or VMS operating systems running X-Windows/OSF Motif. GEC-Marconi Software Systems, Reston, VA 22090. (703) 648-1551.

B.2.7 SPQR/20

SPQR/20 (Software Product, Quality, and Reliability TwentyQuestions) is a software estimation tool distributed by Software Productivity Research (SPR). SPQR/20 is based on the work of Capers Jones [Jones 1986] and incorporates proprietary algorithms. It was one of the first models to use function points as a measure of size to estimate source lines of code. Most of the inputs define experience level, development method, and development environment. Other inputs include project type and class. SPQR/20 estimates maintenance support for up to a five-year period [SPR 1986]. SPQR/20 operates on PC compatible systems. Software Products Research, Inc., Burlington, MA 01803. (617) 273-0140.

B.3 Other Method-Based Software Estimation Tools

This section presents software cost estimation tools that are neither exclusively based on COCOMO nor on Function Point analysis. Several of these models are based on proprietary algorithms and databases.

B.3.1 CA-ESTIMACS

CA-ESTIMACS is part of a family of tools called CA-UNIPACK/PEP marketed by Computer Associates International, Inc. CA-UNIPACK/PEP consists of four tools: CA-ESTIMACS. CA-PLANMACS, CA-ADVISOR, and CA-SuperProject. CA-ESTIMACS uses research drawn from a data base of more than 14,000 completed software projects. The ESTIMACS model is a proprietary model that does not require KLOC as an input, relying instead on "Function-Point-like" measures The original application domain for this model was the insurance industry. Since this model is proprietary, details, such as the equations used, are not available. It is known that, like SLIM and COCOMO, ESTIMACS utilizes a series of 25 questions to adjust model parameters. It develops estimates at or before the requirements definition phase of the software life cycle. This tool allows for early "what-if" analyses of alternative life cycle strategies. A companion tool named CA-FPXpert is also distributed by Computer Associates and is used to estimate the size of the software product. All of these tools operate on PC compatible systems. Computer Associates International, Inc., Calverton, MD 20705. (301) 937-1133.

B.3.2 CEIS

Computer Economics Incorporated Sizing (CEIS) system is marketed by Computer Economics, Inc. Estimation are generated by comparing the attributes of the new project to the attributes of three reference projects of known size. The user determines any six attributes that contribute to the number of lines of code and ranks them in order of importance, then selects three reference projects of known size. Separate algorithms are used to produce four independent estimates and to determine a level of confidence. CEIS works in conjunction with SYSTEM-4. Computer Economics, Inc., Marina Del Rey, CA 90292. (310) 827-7300.

B.3.3 COSTEXPERT

COSTEXPERT is a software estimation tool that uses expert system technology. It was developed by the Institute for System Analysis, Inc. (ISA) and is marketed by Technology Applications/Engineering Corporation. It asks questions about the functionality of the software being developed [ISA 1990]. COSTEXPERT directly estimates software-related efforts such as program management security, and configuration management. It supports multiple languages and different development standards. It also takes into account software reuse. A companion tool named SIZEEXPERT is also distributed by Technology Applications/Engineering Corporation and is used to estimate the size of the software product based on the COSTEXPERT questions. These tools operate on PC compatible systems. Technology Applications/Engineering Corp., Bethesda, MD 20817. (301) 571-8510.

B.3.4 PRICE S

The PRICE S tool is distributed by GE PRICE Systems. This tool was first developed in 1977 primarily for aerospace applications [Freiman and Park 1979]. Equations used by this tool are proprietary, however, descriptions of the methodology scheduling algorithms used can be found in a paper published by GE PRICE Systems [GE Price 1988]. Wolverton [1980] describes the several project and environmental attributes considered in the model's effort adjustment equations. These attributes were chosen specifically for the aerospace applications. As the model is intended for a specific software domain, its use for business and other non-aerospace applications is questionable.

The PRICE S tool is based on Cost Estimation Relationships (CERs) that make use of product characteristics in order to generate estimates. CERs were determined by statistically analyzing completed projects where product characteristics and project information were known. The major input to PRICE S is KLOC. Software size may be input directly or automatically calculated from quantitative descriptions. PRICE S also permits function points to be input as an alternative to KLOC. Other inputs include software function, operating environment, software reuse, complexity factors, productivity factors, and risk analysis factors. Successful use of the PRICE S tool depends on the ability of the user to define inputs correctly. It can be customized to the needs of the user. It is now available for Windows and Unix/Motif. GE Price Systems, Moorestown, NJ 08057. (800) 437-7423.

B.3.5 SASET

The Software Architecture, Sizing and Estimating Tool (SASET) was developed for DOD by the Martin Marietta Corporation. SASET is a forward-chaining rule-based expert system utilizing a hierarchically structured knowledge data base. The data base is composed of projects with a wide range of applications. SASET provides functional software sizing values, development schedules, and associated manloading outputs. It provides estimates for all types of programs and all phases of the development cycle. It also provides estimates for maintenance support and performs a risk assessment on sizing, scheduling, and budget data.

SASET uses a five-tiered approach for estimation including class of software, source lines of code, software complexity, maintenance staff loading, and risk assessment. The user can either input the program size directly or allow SASET to compute size, based on function-related inputs. The tool also has an extensive customization file in which the user can adjust many parameters. It operates on PC compatible systems. Air Force Cost Aanlysis Agency, Arlington, VA 22202. (703) 746-5865.

B.3.6 SEER-SEM

System Evaluation and Estimation of Resources - Software Estimation Model (SEER-SEM) is distributed by Galorath Associates and is under a five year Air Force wide license agreement. It provides software estimations with knowledge bases developed from many years of completed projects [Galorath 1992]. The knowledge base allows estimates with only minimal high level inputs. The user need only select the platform (i.e. ground, unmanned space), application (i.e. command and control, diagnostic), development methods (i.e. prototype, incremental), and development standards (i.e. 2167A). SEER-SEM is applicable to all types of software projects and considers all phases of software development.

A companion tool called SEER-Software Size Model (SSM) is also distributed by Galorath Associates and is used to estimate the size of the software product. SEER-SEM is designed to run on PC compatible systems running Microsoft Windows 3.0/3.1 (Air Force license includes MS-DOS version). It is also available for the Apple Macintosh running

system 6.0.3 and above and the UNIX/SUN workstation. Galarath Associates, Inc., Los Angeles, CA 90009. (310) 670-3404.

B.3.7 SEER-SSM

SEER-SSM is marketed by Galorath Associates and is available to government and contractors under an Air Force-wide contract. It produces software size estimates in lines of code or function points. It also provides its historical database to save time in producing the size estimates. Galarath Associates, Inc., Los Angeles, CA 90009. (310) 670-3404.

B.3.8 SIZE PLANNER

SIZe PLANNER is distributed by Quantitative Software Mangement, Inc. It uses four independent approaches for size estimation including Fuzzy Logic, Function Points, Standard Component, and New/Reuse/Modified sizing. Each approach views the product from a unique perspective. This capabiity provides a cross check for the overall estimate which reduces the uncertainty of the estimate. SIZE PLANNER is used in conjunction with SLIM and operates on PC compatible systems. Quantitative Software Management, Inc., McLean, VA 22102. (703) 790-0055.

B.3.9 SIZEEXPERT

SIZEEXPERT was developed by the Institute for Systems Analysis and is marketed by Technology Application/Engineering Corporation. This tool is an expert judgment tool that produces estiamtes of lines of code based on questions asked by COSTEXPERT. Both tools are packaged and distributed together and operate on PC compatible systems. Technology Applications/Engineering Corp., Bethesda, MD 20817. (301)571-8510.

B.3.10 SLIM

The Software Life Cycle Model (SLIM) is marketed by Quantitative Software Management (QSM). Originally developed from analyses of ground-based radar programs, the SLIM tool has been expanded to include other types of programs. It can be customized for the user's development environment [QSM-SLIM 1987]. SLIM supports all phases of software development, except requirements analysis, as well as all sizes of software projects, but was especially designed to support large projects.

SLIM is a proprietary model; therefore, much of the current model's details are not publicly available. Early publications concerning the model ([Putnam and Fitzsimmons 1979] indicate that the SLIM model uses a KLOC estimate for the software project's general size, then modifies this through the use of the Rayleigh curve model to produce its effort estimates. The user can influence the shape of the curve through two key parameters: the life-cycle development effort (K) and a productivity factor (PF) that accounts for the state of technology. In practice, these values can be chosen by inputting data from completed projects, or by answering a series of 22 questions, from which SLIM will provide recommended values. The SLIM model's central equation is given by

$$\text{KLOC} = (PF)K^{1/3}t_d^{4/3}$$

where PF is the productivity factor, K is the life-cycle effort, and t_d is the development time, and KLOC is the estimated thousands of lines of code. Rearranging the above equation allows one to solve for the development time.

Success in using SLIM depends on the user's ability to customize the tool to fit the software development environment, and to estimate both a Productivity Index (a measure of the software developer's efficiency) and a Manpower Buildup Index (a measure of the software developer's staffing capability). SLIM also provides a life-cycle option which extrapolates development costs into the maintenance phase. A companion tool named SIZE PLANNER is also distributed by QSM and is used to estimate the size of the software product. Quantitative Software Management, Inc., McLean, VA 22102. (703) 790-0055.

B.3.11 SOFTCOST-R and SOFTCOST-ADA

SOFTCOST-R and SOFTCOST-ADA are software estimating tools developed by Reifer Consultants, Inc. (RCI) [Reifer 1989]. SOFTCOST-R is based upon the pioneering modeling work done by Dr. Robert Tausworthe of the Jet Propulsion Laboratory [Tausworthe 81]. It contains a data base of over 1500 data processing, scientific and real-time programs. A key input is KLOC, which can be input directly or computed from Function Points. SOFTCOST-R is applicable to all types of programs, however, it was specifically configured to estimate real-time and scientific software systems, and considers all phases of the software development cycle.

Although the tool's primary input is KLOC, it also uses the same inputs and provides the same outputs as COCOMO which allows direct comparisons to be made. SOFTCOST-R has some unique inputs such as use of peer reviews, customer experience, and degree of standardization. It also supports a standard WBS for task planning and scheduling.

RCI provides SOFTCOST-Ada, which is a tool to estimate Ada and C++ development costs. SOFTCOST-Ada is a cost estimation tool specifically developed to estimate systems using object-oriented techniques. RCI also has a separate size estimation tool called ASSET-R to estimate the size of the software product. SOFTCOST-R, SOFTCOST-Ada, and ASSET-R are leased on an annual license basis, and require a PC compatible running DOS 2.3 or higher. Reifer Consultants, Torrance, CA 90510. (310) 373-8728.

B.3.12 SYSTEM-4

SYSTEM-4 is marketed by Computer Economics, Inc. (CEI). It contains a proprietary model that is based on the work of Jensen, Boehm, Putnam, and other noted software experts [Jensen 1981]. SYSTEM-4 is applicable to all types of programs and all phases of the software life cycle. Inputs consist of size (KLOC), twenty environmental factors, seven development factors, software type, and constraints. This tool comes with 23 pre-defined default parameter files. The default sets provide typical values for all parameters except size. There are also seven parameter subset files for various implementations of DOD-STD-2167A, and varying degrees of Ada experience. CEI has a companion software size estimating tool, CEIS. These tools operate on PC compatible systems. Computer Economics, Inc., Marina Del Rey, CA 90292. (310) 827-7300.

B.4 References for Appendix B

Barrow, D., S. Nilson, and D. Timberlake. *Software Estimation Technology Report*, Software Technology Support Center (STSC), Hill AFB, UT, 1993.

Freiman, F. R. and R.E. Park. "Price Software Model - Version 3: An Overview," Proceedings, *IEEE - Workshop on Quantitative Software Models*, IEEE Catalog No. TH0067-9, 32-41, October 1979.

Galorath Associates. *SEER User's Manual*, Los Angeles, CA, 1989.

GE Price Systems, "PRICE S User's Manual," Moorestown, NJ, Price Systems 1989.

ISA Corporation. *ISA Cost Expert User's Guide*, Technical Applications/Engineering Corp., Bethesda, MD. 1990

Jensen, R. W. "A Macro-level Software Development Cost Estimation Methodology," *Proceedings of Fourteenth Asilomar Conference on Circuits, Systems and Computers*, IEEE, New York, 1981

Jones, C. *Programming Productivity*, New York, McGraw-Hill, 1986

Jones, C. *Applied Software Measurement; Assuring Productivity and Quality*, McGraw-Hill, New York, NY. 1991

Kyle, R.A. *REVIC Software Cost Estimating Model User's Manual, Version 9.0*, Air Force Contract Management Division, Albuquerque, NM, Feb. 1991.

NASA/JSC. *COSTMODL User's Guide Version 5.2*, NASA Johnson Space Center, Houston, TX, , January 1991.

Price Systems. "The Central Equations of the Price Software Cost Model," *Proceedings of the Fourth Annual COCOMO User's Group Meeting*, Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA, Nov. 1988.

Putnam, L.H. and A. Fitzsimmons. "Estimating Software Costs," *Datamation*, Sept.-Nov. 1979.

Putnam, L., and W. Myers. *Measures for Excellence: Reliable Software, On Time, Within Budget*, Yourdon Press, 1992.

QSM Corporation, *Slim User's Manual,* McLean, VA, 1987.

Reifer, D.J. *SOFTCOST-R User's Manual, Version 8.0*, Reifer Consultants, Torrance, CA, 1989.

Software Productivity Research (SPR), Inc., *SPQR/20 User's Guide*, Cambridge, MA, 1986.

Tausworthe, R.C. *Deep Space Network Cost Estimation Model*, JPL Publication 81-7, Jet Propulsion Laboratory, April 1981.

Wolverton, W.R. "Airborne Systems Software Acquisition engineering Guidebook: Software Cost Analysis and Estimating," U.S. Air Force ASD/EN, Wright Patterson AFB, OH, FEb. 1980.

Appendix C
The f_2 and f_3 Conditional Expected Values of the Triangular Distribution

This appendix includes the derivation of the f_2 and f_3 conditional expected value equations for the triangular distribution. Although not commonly used in risk-based decision making situations, these equations are included for completeness and for use in the limited scenarios where they are applicable.

C.1 The High-probability, Low-damage Expected Value, f_2

Asbeck and Haimes [1984] define the high-probability, low-damage expected value f_2, as

$$f_2 = \frac{\int_{-\infty}^{\beta} x f(x)dx}{\int_{-\infty}^{\beta} f(x)dx}. \tag{C.1}$$

Deriving the f_2 conditional expectation for the triangular distribution is accomplished by substituting the element corresponding to $a \le \beta \le c$ of Eq. (4.1) into Eq. (C.1):

$$f_2 = \frac{\int_{a}^{\beta} x f(x)dx}{\int_{a}^{\beta} f(x)dx} = \frac{\int_{a}^{\beta} 2x(x-a)dx}{\int_{a}^{\beta} 2(x-a)dx}$$

$$= \frac{\left[\frac{2}{3}x^3 - ax^2\right]_a^{\beta}}{\left[x^2 - 2ax\right]_a^{\beta}} = \frac{\left[\frac{2}{3}\beta^3 - a\beta^2 - \frac{2}{3}a^3 + a^3\right]}{\left[\beta^2 - 2a\beta - a^2 + 2a^2\right]} = \frac{\frac{2}{3}\beta^3 - a\beta^2 + \frac{1}{3}a^3}{\beta^2 - 2a\beta + a^2}$$

$$= \frac{(a^3 - 3a\beta^2 + 2\beta^3)}{3(a-\beta)^2} = \frac{(a-\beta)^2(a+2\beta)}{3(a-\beta)^2} = \frac{a+2\beta}{3}.$$

Thus,

$$f_2 = \frac{a+2\beta}{3}, \quad a \le \beta \le c. \tag{C.2}$$

C.2 The Moderate-damage, Moderate-probability Conditional Expected Value, f_3

Finally, we derive the intermediate conditional expectation, f_3, for the triangular distribution. This expected value is similar to the unconditional expected value in that it also commensurates events of low probability and high damage with those of high probability and low damage. The general form of the f_3 conditional expected value is [Asbeck and Haimes 1984]:

$$f_3 = \frac{\int_{\beta_1}^{\beta_2} xf(x)dx}{\int_{\beta_1}^{\beta_2} f(x)dx}, \text{where } \beta_1 \le \beta_2. \tag{C.3}$$

In deriving the f_3 conditional expected value for the triangular distribution, we consider the two partitioning points β_1 and β_2, such that $a \le \beta_1 \le c \le \beta_2 \le b$. Substituting Eq. (4.1) into Eq. (C.3) requires solving

$$f_3 = \frac{\left(\int_{\beta_1}^{c} xf(x)dx + \int_{c}^{\beta_2} xf(x)dx\right)}{\left(\int_{\beta_1}^{c} f(x)dx + \int_{c}^{\beta_2} f(x)dx\right)}.$$

Solving the numerator of the above expression produces

$$\frac{1}{(b-a)(c-a)}\int_{\beta_1}^{c} 2x(x-a)dx + \frac{1}{(b-a)(b-c)}\int_{c}^{\beta_2} 2x(b-x)dx$$

$$= \frac{1}{3(b-a)(c-a)}\left[2c^3 - 3ac^2 - 2\beta_1^3 + 3a\beta_1^2\right] + \frac{1}{3(b-a)b-c)}\left[2c^3 - 3bc^2 - 2\beta_2^2 + 3b\beta_2^2\right],$$

while solving the denominator results in

$$\frac{1}{(b-a)(c-a)}\int_{\beta_1}^{c} 2(x-a)dx + \frac{1}{(b-a)(b-c)}\int_{c}^{\beta_2} 2(b-x)dx$$

$$= \frac{1}{(b-a)(c-a)}\left[c^2 - 2ac - \beta_1^2 + 2a\beta_1\right] + \frac{1}{(b-a)(b-c)}\left[2b\beta_2 - \beta_2^2 - 2bc + c^2\right].$$

Combining the numerator and denominator results, and without much simplification other than canceling the obvious (*b*-*a*) terms, we observe

$$f_3 = \left(\frac{\left[\frac{(2c^3 - 3ac^2 + 3a\beta_1^2 - 2\beta_1^3)}{3(c-a)} + \frac{(2c^3 - 3bc^2 + 3b\beta_2^2 - 2\beta_1^2)}{3(b-c)}\right]}{\left[\frac{(c^2 - 2ac + 2a\beta_1 - \beta_1^2)}{(c-a)} + \frac{(c^2 - 2bc + 2b\beta_2 - \beta_2^2)}{(b-c)}\right]} \right), \quad a \le \beta_1 \le c \le \beta_2 \le b. \tag{C.4}$$

Equations (C.2) and (C.4), coupled with Eqs. (4.3) and (4.6) are the exact-form conditional expectation equations for the triangular distribution.